

500

Damir Muraja

ORAO

UVOD U RAD I PROGRAMIRANJE

Narodna tehnika SR Hrvatske

Zagreb, 1985.

SADRŽAJ

UVOD	3	ČUVANJE PROGRAMA	49
ŠTO MOŽE -ORAO-	5	POTPROGRAMI	51
PRIKLJUČIVANJE	5	PODACI UNUTAR PROGRAMA	55
KAKO POČETI	7	NUMERICKE FUNKCIJE	58
TASTATURA	8	STRING-FUNKCIJE	61
NEKI „STRUČNI“ POJMOVI	10	PRINT-FUNKCIJE	65
ŠTO JE BASIC	11	KORISNICKE FUNKCIJE	69
PRIMJENA BASICA	12	POLJE PODATAKA	70
PA, DA POČNEMO!	13	LOGICKE OPERACIJE	73
PISANJE TEKSTA	16	LOGICKE OPERACIJE U BASICU	77
KOMBINACIJE IZRAZA U		GRAFIKA	79
PRINT-NAREDBAMA	18	MEMORIJA	82
MOJ PRVI PROGRAM	19	POZIVI STROJNIH POTPROGRAMA	86
ISPRAVLJANJE TEKSTA		MONITOR	88
PROGRAMA (EDITOR)	23	ŠTO DALJE?	93
PETLJA	25	PODACI	94
UNOS PODATAKA (INPUT)	29	Popis naredbi u BASICU	94
NAPOMENE U PROGRAMU (REM)	32	Set i kodovi znakova (ASCII)	95
DONOŠENJE ODLUKA (IF)	34	Popis grešaka	96
NAREDBE ZA PREKIDANJE		Popis naredbi za mikroprocesor 65 02	96
PROGRAMA	38	PROGRAMI	98
FUNKCIJA INT	39	Preračunavanje brojeva	98
SLUČAJNI BROJ (RND)	41	Program za crtanje	101
PISANJE PROGRAMA	43	Porez (igra)	104
Opis i pravila igre	43	Generiranje zvuka	107
Dijelovi programa	43		
Skica programa	44		

UVOD

Mikroračunalo »orao« namijenjeno je, u prvom redu, za učenje programiranja. To ne znači da se tim mikroračunalom ne mogu obavljati i neki teži zadaci. Ova knjiga napisana je s namjenom da posluži kao priručnik iz kojeg se može naučiti BASIC kojim »orao« raspolaže.

Ne treba se zanositi idejom da se programiranje može naučiti samo čitanjem knjige. Da biste naučili programirati morate provesti mnogo sati za računalom. Ova knjiga će vam poslužiti da naučite instrukcije kojima »orao« raspolaže i njihovu sintaksu. Nakon toga preostaju vam mnogi sati rada za mikroračunalom i učenje po metodi pokušaja i pogreške. Greške pri radu neka vas ne plaše jer jedino iz takvih grešaka može se »stvarno« shvatiti rad računala.

Jednom je netko lijepo rekao da je osnovna mana računala to što rade što im naredimo, a ne ono što želimo. Kad budete napisali dvadesetak programa i kad ispravile greške koje će se u njima javiti, postat će vam jasna ova izreka.

ŠTO MOŽE «ORAO»

Kao što sam već rekao, osnovna namjena tog mikro-računala je učenje programiranja. Mikroračunalo «orao» raspolaže grafikom srednje rezolucije (256 x 256 točaka), programibilnim generatorom zvuka, mogućnošću rada s magnetskim medijima i dakako, setom slova koji obuhvaća i naša latinična slova. Svakim od ovih svojstava mikroračunala «orao» baviti ćemo se u zasebnom poglavlju, a kada iskoristimo sve njegove mogućnosti, vidjet ćemo da je pred nama mikroračunalo koje se može i te kako korisno upotrijebiti.

Pogledajmo malo neke karakteristike mikroračunala «orao». Ako vam ovi podaci ništa ne znače, slobodno ih preskočite. Kasnije, kada naučite više o programiranju sve to za vas će dobiti jasno značenje, pa ćete tek onda pročitati i ovaj dio.

«Orao» raspolaže aritmetikom kliznog zareza (floating point). Računa se s točnošću od šest znamenki. Brojevi su prikazani u eksponencijalnom obliku: najmanji pozitivni broj je 10E-38, a najveći pozitivni broj 10E38.

Dužina stringova je najviše 255 znakova.

Najveća dužina programske linije je 72 znaka. Može je definirati i manja gornja granica za dužinu programske linije.

Broj programske linije je pozitivni cijeli broj veći od 0 ili manji od 64000.

Računalo posjeduje BASIC INTERPRETER koji podržava rad s kasetofonom i ispis na RS232 port.

Moguće je pozivanje strojnih programa iz BASICA.

Računalo je konstruirano oko mikroprocesora 6502 i posjeduje monitor i miniassembler koji omogućuju programiranje u mašinskom jeziku. Ovisno o verziji, računalo posjeduje 32 ili 16 K RAM-a.

Ekran je organiziran kao bitna mapa i ima 256 x 256 točaka. Ne postoji tekst-ekran i sav se tekst ispisuje u grafičkom ekranu u matrici 8 x 8, odnosno 32 reda sa po 32 znaka u redu.

PRIKLJUČIVANJE

Da bi «orao» mogao raditi, potrebno je da ga povežemo na mrežu preko mrežnog kabela koji se nalazi

na njegovoj poleđini. Osim spoja na mrežu potrebno je »orla« povezati s televizorom ili s videomonitorom. Za povezivanje služe predviđeni priključci (vidi sliku 1). Ako »orla« povežemo s TV-prijemnikom potrebno je pravilno podesiti TV-prijemnik. To ćemo učiniti tako da ga prebacimo na »prvi program« i sliku potražimo oko devetog kanala. Kada nađemo pravi kanal, na ekranu moramo ugledati ovu sliku:

*** ORAO ***

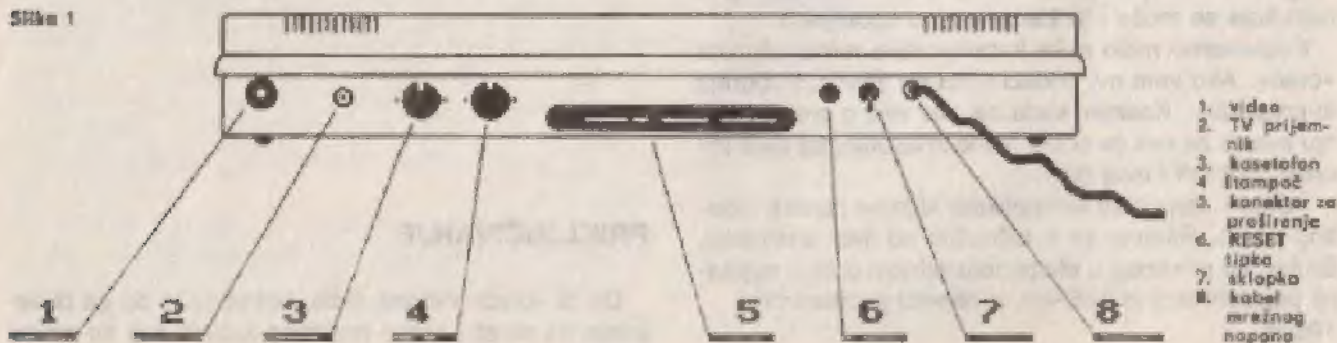
To je znak da je »orao« spreman za rad. Ako imate problema s pronalaženjem slike ispitajte da li je »orao« uopće uključen (prekidač mrežnog napona na poleđini, vidi sliku 1), zatim da li ste pravilno povezali »orao« i televizor. Dosta je česta greška da se antenski kabel

umjesto u antenski priključak u »orlu« priključi na priključak za videosignal. Tada nećemo dobiti sliku na televizoru. Ako sve pravilno spojite, dobit ćete na ekranu sliku koju je možda potrebno još malo »dotjerati«. To ćete postići potencijometrima za regulaciju svjetline i oštrote slike na televizoru. Potrebno je postići tamnu pozadinu (gotovo crnu) sa svijetlim slovima na kojima se jasno razaznaju točkice od kojih su slova sastavljena.

Pravilna podešenost slike je jako važna za rad jer mutna slika ili slika koja podrihtava opterećuje oči pa rad pri takvoj slici nije zdrav. Može vam se desiti da posljednji red slike izlazi izvan granica ekrana. To, dakako, onemogućuje rad pa je potrebno na potencijometrima za regulaciju visine slike (ti se u pravilu nalaze na poleđini televizora) podesiti visinu slike.

Na kraju, da budemo pošteni, moramo reći da »orao« može nesmetano raditi i bez televizora. Televizor je tu

Slika 1



zbog toga da vidimo rezultate programa koje »orao« izvršava i bez njega ne možemo komunicirati s »orlom«.

Osim televizora na »orao« možemo priključiti i kasetofon u koji ćemo pohranjivati programe i štampač na kojem ćemo ispisivati rezultate našeg rada. O tome ćemo više u poglavljima koja se bave radom sa kasetofonom.

Bolji poznavatelji elektronike mogu preko vanjskog konektora povezati i neke druge sklopove na mikroračunalo »orao« (na primjer, releji za kontrolu nekih procesa).

KAKO POČETI

Kada smo obavili sve potrebno i pravilno priključili mikroračunalo »orao«, dobili smo ispod naslova »orao«

zvjezdicu iza koje se nalazi treptava crtica. Ova crtica naziva se kursor (cursor). Kursor nam pokazuje gdje će se ispisati slijedeći znak koji ukucamo s tastature. Svaki put kad ugledamo ovu blještavu crticu računalo očekuje da otipkamo neku naredbu ili podatak.

Zvjezdica koja se nalazi ispred kursora govori nam da se »orao« nalazi u monitoru (više o monitoru u posebnoj poglavlju). Kako želimo raditi u BASICU, potrebno je kontrolu računala prebaciti u BASIC. Da to jednostavnije kažemo, potrebno je »uključiti« BASIC. To ćemo postići tako da otipkamo BC. Na ekranu ćemo ugledati:

*BC

Nakon toga pritisnete tipku (CR). Na ekranu će se pojaviti

MEMORIJA ?

Ovdje je dovoljno da samo pritisnete (CR). Ispisat će se:

DULJINA LINIJE ?

Ponovno pritisnete (CR). Sada će »orao« ispisati slobodni memorijski prostor. Na ekranu će pisati:

23551 LOKACIJA ili
7167 LOKACIJA

To ovisi o veličini memorije koja je ugrađena u vašeg »orao«. »Orao« se pojavljuje u dvije varijante: sa 32 K ili sa 16 K memorije. U ovisnosti koju varijantu mikro-računala imate, dobit ćete i komentar o raspoloživoj memoriji. Ispod obavijesti o slobodnoj memoriji ispisat će se:

>

Prvi znak naziva se PROMPT i govori nam da je »orao« u BASICU. Ovaj znak pojaviti će se svaki put kada računalo čeka da unesemo neku naredbu ili programsku liniju. Kada računalo izvršava program pojava ovog znaka signalizirat će nam da je program izvršen ili da je nastupio prekid.

Kao što sam već rekao, blještava crtica je kursor i označava mjesto gdje će se upisati tekst koji otipkamo na tastaturi.

Prije nego nastavimo s analizom rada s računalom treba napomenuti da nije moguće oštetiti (pokvariti)

računalo tipkanjem na tastaturi. Bez obzira na to što ukucali, uvijek možemo isključiti računalo i potom ga ponovno uključiti i početi od početka. Zbog toga možete slobodno iskušati sve što vam padne na pamet bez ikakva straha.

TASTATURA

Pogledajmo malo tastaturu mikroračunala »orao«. Oni koji su već sjedili za tastaturom pisaćeg stroja primijetiti će veliku sličnost između ove dvije tastature. Na lijevoj strani nalazi se velika skupina tipki na kojima su sva slova naše abecede uključujući i slova Q, W, X, Y. Raspored je u skladu s jugoslavenskim standardom za raspored slova i nazivamo ga QWERTZ prema slovima u drugom redu tastature. Osim slova na tastaturi vidite i brojeve koji su smješteni u prvom redu. Vidimo da među brojevima postoje i brojevi 1 i 0 kojih

na pisaćim strojevima nema. U pisanju na računalu nije dozvoljeno pisati malo slovo l umjesto broja 1, odnosno, veliko slovo O umjesto broja 0. Da bi se nula lakše razlikovala od slova O preko nje je povučena kosa crta pa podsjeća na grčko slovo fi.

Na nekim tipkama upisana su po dva znaka. Donje znakove dobivamo jednostavnim pritiskom na tipku. Da biste napisali gornje znakove morale osim tipke pritisnuti i tipku SHIFT (to su one dvije tipke na kojima ništa ne piše). Svejedno je koju od tipki SHIFT pritisnete, ali je potrebno pritisnuti prije tipke na kojoj je znak koji želimo napisati i držati je sve dok se znak ne ispiše. Tipka SHIFT ima još jednu funkciju. Kad radite s uključenim malim slovima, pritiskom na ovu tipku i bilo koje slovo dobit ćete veliko slovo.

Osim razmaknice (široka tipka bez oznaka) koja služi za pisanje razmaka između riječi, u ovoj skupini tipki vidite još i dvije tipke označene slovima CR odnosno CTL. Tipka (CR) služi da računalu javite da ste završili unošenje podataka ili naredbe. Nju morate pritisnuti svaki put kad unesete bilo kakve naredbe. Tek kad je pritisnete, računalo će razmotriti ono što ste napisali i ako je moguće izvršiti vaš nalog.

Tipka (CTL) služi za specijalne funkcije i upotrebljava se u paru s nekim slovom, slično kao i SHIFT. Ovdje vidite popis svih kontrolnih funkcija:

- (CTL) L brisanje ekrana
- (CTL) G zvučni signal
- (CTL) F briše tekst od kursora do kraja ekrana

- (CTL) H pomiče kursor ulijevo
- (CTL) K pomiče kursor prema gore
- (CTL) I pomiče kursor udesno
- (CTL) J pomiče kursor prema dolje
- (CTL) V uključuje zvučni signal prilikom tipkanja
- (CTL) E briše tekst od kursora do kraja linije
- (CTL) B uključuje štampač
- (CTL) U isključuje štampač
- (CTL) D vraća kursor na početak ekrana
- (CTL) M vraća kursor na početak reda
- (CTL) C prekida izvođenje programa

Od svih ovih funkcija najčešće ćete upotrebljavati (CTL) C kojom prekidamo izvođenje programa.

Na desnoj strani računala vidite dvije skupine po četiri tipke. Gornju skupinu tvore tipke označene strelicama koje služe za pomicanje kursora u sva četiri smjera. To vam je potrebno prilikom editiranja o čemu će još biti govora.

Donje četiri tipke PF1, PF2, PF3 i PF4 su funkcijske tipke. Značenje ovih tipki je ovo:

- (PF1) prebacuje iz malih slova u velika i obrnuto
- (PF2) služi za uključivanje štampača
- (PF3) uključuje odnosno isključuje inverzni ispis slova
- (PF4) kopira tekst ispod kursora

Tipka (PF4) detaljnije je objašnjena u poglavlju u kojem se govori o editiranju.

Tipkama na tastaturi pridružena je tzv. autorepeat-funkcija. To znači ako neku tipku malo duže držite, počet će automatsko ispisivanje istog znaka.

NEKI »STRUČNI« POJMOVI

U ovom poglavlju objašnjeni su neki od termina koje ćemo češće sretati u nastavku teksta.

RAM – To je kratica od Random Access Memory, što možemo prevesti kao memorija sa slobodnim pristupom. U ovu memoriju upisujemo svoj program, u njoj se nalazi slika, u nju računalo sprema međurezultate itd. Sadržaj RAM-a gubi se nakon što se računalo isključuje.

ROM – Kratica od Read Only Memory, što znači memorija koju možemo samo čitati. Unutar ove memorije nalazi se program koji omogućuje računalu da ra-

zumije naše programe, čita tastaturu, crta sliku itd. Sadržaj ove memorije upisan je u tvornici prilikom izrade računala i ne možemo ga mijenjati (zato i kažemo da je to memorija samo za čitanje). Sadržaj ROM-a gubi se nakon što isključimo računalo.

K – Vjerojatno ste već primijetili da kad govorimo o veličini memorije iza broja stavljamo veliko slovo K. Kažemo, na primjer, da »orao« sadržava 16K ROM-a. Slovo K mnogi čitaju kao »kilo«, ali to nije točno jer prefiks »kilo« znači tisuću (npr. kilometar je tisuću metara). Kad govorimo o računalima, K označava 1024, odnosno dva na desetu. To znači da kad kažemo 16K ROM-a mislimo zapravo na 16×1024 , odnosno 16384 bajta ROM-a. U cijelom daljnjem tekstu oznaka K označava 1024 bajta.

BIT – Najmanja informacija kojom barata računalo. Riječ bit kratica je od Binary digIT što znači binarna znamenka. Bit može imati vrijednost jedan ili nula.

BYTE – (u daljnjem tekstu bajt). To je osnovna jedinica kompjutorske memorije. Bajt je skup od osam bitova (jedinica ili nula). U njemu može biti vrijednost od 0 do 255, odnosno od nula do dva na osmu minus jedan. Kada kažemo, da računalo ima 16K bajta memorije, to znači da može zapamtiti 16384 brojeva u rasponu od 0 do 255. Da pojednostavimo možemo reći da je svaki bajt jedno slovo, odnosno da računalo može zapamtiti 16384 slova što odgovara tekstu od 9 stranica.

ŠTO JE BASIC

Vec sam u nekoliko navrata rekao da mikroračunalo »orao« radi u BASICU – da ću u ovoj knjizi najviše govoriti o BASICU. Pogledajmo zato što je BASIC.

Ljudi se u međusobnom komuniciranju služe govorom. Pri tome postoje određeni jezici (na primjer, hrvatski ili engleski jezik). Svi ljudi koji poznaju određen jezik međusobno se razumiju. Ako nekome tko poznaje samo jedan jezik kažemo nešto na nekom drugom jeziku, on nas neće razumjeti.

Kao i ljudi i računala razumiju jedan jezik. Jedan jezik na računalu obuhvaća skup riječi koje tvore taj jezik. Jedan od takvih jezika je i BASIC. Kada kažemo da neko računalo »razumije« BASIC, to znači da to računalo razumije skup riječi koje zajedno sačinjavaju jezik BASIC. Da bismo ove jezike razlikovali od jezika kojima govore ljudi, nazivamo ih programski jezici.

Među ljudima koji govore određenim jezikom, ma manjih ili većih razlika između riječi koje oni upotrebljavaju – u ovisnosti o kraju iz kojeg potječu. Takve podskupove jezika nazivamo dijalektima.

Sva računala jednog tipa razumiju isti BASIC pa možemo program napisan na jednom »orao« prenijeti na drugi »orao«. Na računalima različitih proizvođača postoje manje ili veće razlike između programskih jezika. Tako program napisan u BASICU na mikroračunalu »spectrum« neće raditi na mikroračunalu »orao« sve dok se ne izvedu potrebne izmjene u programu. Zbog toga govorimo o dijalektima programskih jezika. Ova knjiga bavi se dijalektom programskog jezika BASIC za mikroračunala »orao«.

Za razliku od jezika u svakodnevnom govoru za čije učenje je potreban višegodišnji rad, programski jezike možemo naučiti u vrlo kratkom vremenu. To slijedi iz toga što programski jezici sadržavaju vrlo malen skup riječi (stotinjak). Kada naučimo jedan dijalekt nekog programskog jezika, potrebno nam je samo nekoliko sati da svladamo drugi dijalekt istog programskog jezika. Isto tako, ako dobro poznajemo jedan programski jezik, vrijeme potrebno za učenje svakog sljedećeg bit će sve kraće i kraće.

Programski jezik BASIC razvijen je u Sjedinjenim Američkim Državama prije dvadesetak godina. Prilikom njegova stvaranja osnovna ideja bila je da se stvori jezik opće namjene. Zbog toga je ovaj program-

ski jezik jednostavan za učenje i lako primjenljiv na bilo koju vrstu problema.

«Orao» kao ni sva ostala mikroračunala ne razumije naredbe BASICA već ih mora «prevoditi» u instrukcije strojnog jezika. To «prevođenje» je prilično spor proces zbog toga su programi pisani u BASICU znatno sporiji od programa pisanih u strojnom jeziku. Ne dajte da vas riječi «sporiije» dovede u zabludu jer računalo će u BASICU mnoge zadatke obavljati jako brzo.

Prema načinu prevođenja razlikujemo dvije vrste rada s BASICOM. Prva je interpretiranje, odnosno prevođenje svake instrukcije neposredno prije nego što se izvrši. BASIC-e koji lako rade nazivamo interpreteri. Druga vrsta je kompajliranje, odnosno prevođenje čitavog programa prije nego što počne njegovo izvršavanje. Takvi BASICI nazivaju se kompajleri. Kompajleri daju programe koji se znatno brže izvršavaju ali ipak još uvijek sporije od programa pisanih u strojnom jeziku. Međutim, rad s interpreterima je znatno lakši jer je moguće lakše pronalaženje i ispravljanje grešaka i unošenje izmjena u program. (Kompajlirani programi praktično se ne mogu mijenjati.) Zbog toga je rad sa interpreterom jedini pogodan za početnika. Mikroračunalo «Orao» posjeduje interpreter BASICA.

Rekao sam već da je BASIC razvijen u SAD (kao i većina drugih programskih jezika). Radi toga su sve riječi koje postoje u BASICU uzete iz engleskog jezika. Poznavanje engleskog jezika nije preduvjet za rad u BASICU jer je dovoljno znati šta koja komanda radi.

Oni koji uče engleski jezik svakako će lakše zapamtiti pojedine komande na temelju njihovog značenja. U tablici 1 nalaze se sve BASIC-naredbe. O svakoj od njih bit će kasnije više govora.

PRIMJENA BASICA

U prethodnom poglavlju rekli smo da je BASIC relativno spor jezik. To treba shvatiti uvjetno jer je «oruđe» potrebno manje od tri sekunde da broj od 1 do 1000 u krugovima «stručnjaka» za mikroračunala vrlo često čete čuti da je basic loš, spor, nestrukturiran itd. Sv oni imaju neki svoj «najbolji» jezik i svoje «najbolje» računalo. Ne obazirite se na te primjedbe jer ti isti «stručnjaci» jako nešto znaju i uči su osnovne programiranja baš u BASICU.

U BASICU su napisani mnogi ozbiljni programi. Na primjer, tekst-procesor (programi za obradu teksta),

programi za unakrsnu analizu brojeva, programi za crtanje itd. Ima dosta igara koje su napisane u BASICU, a mnoge od njih postale su zavidan komercijalni uspjeh.

OVO JE GLUPOST

Kada pritisnete (CR) računalo će razmotriti to što ste napisali, a kako je to za njega zaista glupost, ispod toga će napisati:

? BN GREŠKA

>

--

Time vas je «orao» obavjestio da ne razumije to što ste napisali, a zatim ponovno postavio PROMPT i kursor spreman da izvrši vaše naredno naređenje.

Napišimo sada nešto pametnije. Upišite

PA, DA POČNEMO!

Nakon što ste uključili računalo i ušli u BASIC, «orao» od vas očekuje naređenje šta da radi. Ako otkucate bilo što i pritisnete (CR), računalo će razmotriti što ste mu naredili i ako je moguće, to će i izvršiti. Na ekranu treba da imate PROMPT (>) i ispod njega kursor (_). Ako to nije tako, počnite od početka. Pritisnite tipku na porednim računala i ponovite postupak pozivanja BASICA. Sada napišite

PRINT 1

Računalo će u narednom redu ispisati 1, a zatim ponovno PROMPT i kursor. Riječ PRINT je prva BASIC-naredba koju smo upoznali. PRINT na engleskom znači «stampaj», a to isto znači i ova BASIC-naredba. Kada napišemo PRINT 1 naređujemo računalu da napiše broj 1. «Orao» to razumije i zato odmah izvršava naše naredbe i ispisuje broj 1 u narednom redu.

Napišite sada:

PRINT 3+4

Nakon što pritisnete (CR) «orać» će u narednom redu napisati 7. Vidimo da nije napisao $3 + 4$, već je izračunao izraz i napisao rezultat. Pod pojmom izraz podrazumijevamo bilo koji skup brojeva povezanih matematičkim operatorima. Možete napisati ove naredbe:

PRINT 7-2

PRINT 8/4

PRINT 2*3

Svak od ovih izraza «orać» će izračunati i ispisati vam rezultat. «Orac» raspoznaje ove matematičke operatore:

- + za zbrajanje
- za oduzimanje
- * za množenje
- / za dijeljenje

Vidimo da se za množenje upotrebljava zvjezdica, a ne znak (x), niti točka kako ste možda navikli. Također, nije potrebno upotrijebiti dvije točke (:) kao znak za dijeljenje.

Napišite sada **PRINT 3/2**. «Orac» će napisati rezultat 1.5. Vidimo da umjesto zareza u decimalnim brojevima služi decimalna točka. To morate zapamtiti! Jer će «orać» ako napisete 7.3 to shvatiti kao dva broja (sedam i tri), a ne kao decimalni broj.

«Orac» pazi na prioritet matematičkih operacija. To znać da će se prvo izvršiti matematičke operacije višeg prioriteta (množenje i dijeljenje), a tek potom operacije nižeg prioriteta (zbrajanje i oduzimanje). Napišite, **PRINT 3 + 2 * 4**. «Orac» će najprije pomnožiti 2 i 4, a potom pribrojiti 3 i ispisati rezultat 11. Ako želimo promijeniti redosjed izvršavanja operacija, možemo upotrijebiti zagrade. Napišite:

PRINT (3+2)*4

na ekranu će se ispisati rezultat 20. Dozvoljeni su i složeniji izrazi, na primjer:

PRINT (3+2)*(4-3)/(7-1)+1)

Kada trebamo više ugnježđenih zagrada (jedne unutar drugih) uzimamo uvijek okrugle zagrade, a ne uglate ili vidičaste kako ste učili u matematici. Osim matematičkih operacija zbrajanja, množenja, dijeljenja i oduzimanja postoji i operator za potenciranje. To je strelica prema gore. (Kupnje desna tipka u drugom redu tastature, a ne tipka z skupine tipaka sa strelicama). Kada pritisnete ovu tipku, na ekranu će se ispisati okrenuto malo slovo v. Potenciranje ima najviši prioritet i izvodi se prije svih ostalih operacija, osim ako zagrade ne određuju drugaćije.

Sve što sam do sada opisao može i svak "prosečan" kalkulator. Ali bez stalnog upisivanja naredbe PRINT. Pogledajmo sada nešto što "orao" može, a kalkulator ne može. Napišite

```
A=3
PRINT A
```

Na ekranu će se ispisati broj 3. Znak jednako (=) ima na mikračunalima posebno značenje pridruživanja vrijednosti. Kada napišete A=3, u memoriji računala pamti se broj 3 kojemu je pridruženo ime A. Ovako zapravo i izračunak nazivamo variabla.

Varijable je prošlost u memoriji i komet je p... je eno neko ime i vrijednost. Ime varijable sastoji se od dva z... a drugo k... pi... mora bi... slovo engleske abecede (mislim, dozvoljena naša slova a, b, c, d, s, ž, a drug... znak može biti ili slovo engleske abecede ili broj. Kao što ste vidjeli u primjeru, ime varijable može biti i samo jedno slovo.

Nakon što varijablu jedinstveno definiramo, možemo se s njom poslužiti u izrazima. Možete napisati

```
PRINT A+2
```

I "orao" će napisati rezultat 5 iako ste prethodno de-

finirali A=3. Vrijednost pojedine varijable može se neograničeno puta mijenjati (otud i njegovo ime), prema tome možete napisati

```
A=5
PRINT A
A=7
PRINT A
```

Vidimo da je naredba PRINT A u prvom primjeru ispisala rezultat 5, a u drugom 7. Ako neku varijablu nismo definirali, a zahtijevamo od računala da nam ispiše njenu vrijednost, dobit ćemo rezultat 0. Možemo reći da sve varijable prije nego što su definirane imaju vrijednost nula.

Vrijednost varijable može se izračunavati na primjer

```
X = 7+4
PRINT X
```

Ovdje u izrazu je znakovita vrijednost izraza s desne strane, a za nju se dobivena vrijednost pridružuje varijabli X. Moguće je i izraz

```
A=5
X=A*2
PRINT X
```

Sada smo u definiciji varijable X upotrijebili prethodno definiranu varijablu A . Pri tome se treba držati pravila da se s lijeve strane znaka jednakosti nalazi samo varijabla koju definiramo a s desne strane se mogu nalaziti bilo kako složeni izraz u kojima imamo jednu ili više varijabli. Potrebno je jedino da su sve varijable koje upotrebljavamo s desne strane znaka jednakosti već prethodno definirane.

Jedan od klasičnih primjera primjene varijabli je definiranje broja π . Ako morate izračunavati više izraza u kojima se spominje π , napisite

```
PI=3.141
```

Nakon toga možete površinu kruga koji ima polumjer 5 izračunati ovako:

```
PRINT 5*5*PI
```

Možete to napisati i ovako

```
R=5  
P=R*R*PI  
PRINT P
```

Vjerujem da ste primjetili da se u prethodnom primjeru kvadrat broja R računa izrazom $R*R$. Umjesto toga

mogli ste napisati potenciranje i eksponent 2, ali je prvi postupak brži i točniji

PISANJE TEKSTA

Osim rada s brojevima «*prao*» može raditi i sa slovima. Za pisanje slova služi nam naredba **PRINT** kao i za brojeve. Da bi «*prao*» ispisao neki tekst, on mora biti u navodnicima. Upišite ovu liniju

```
PRINT "ZDRAVO, JA SAM PRAO."
```

Na ekranu ćete ugledati ispis teksta koji ste naveli u navodnicima. Takav tekst u kompjutorskoj terminologiji naziva se *string*. Unutar navodnika može se nalaziti

matrat da le tekst završen.

İlde i razmaka

[illegible]**Možete napísať:**

```

AS="DUHAR DAN"
PRINT AS

```

de intrare în rețea: DDRAR DAN. Svakom korisniku pripisano je [HINT] Adresa koja se ispisat tek nakon što se odredi2) string-variabli AS.

On distinguishes three variables: *mezzo* (medium), *modo* (mode) and *motus* (motion). *Mezzo* is the variable that is the most important in the theory of the *mezzo* (medium).

```
B$="ZDRAW"
```

1. Original 35MM film of each set
2. 16MM DUBBED copy

```
LIB=" DAN"
LIBTYPE
LIBINT C#
```

to u primjeru:

```
A$="JA SAM"  
B$=A$+"OKAD"  
PRINT B$
```

Ako neki string varijablu niste definisali a naredbe računari da je spise računari neće ispisati niti string varijabla koje niste definisali (majuskužno Nije zbir svih razar string Ako zelite neka string varijablu -obrisati- možete nap sati:

As — " "

Pri upisivanju naredbi u računalo možemo stavljati razmake na bilo koja mjesta. Ovi zrazi su potpuno jednaki:

```
P R I N T 1
PRIN      T      1
P R   I N   T 1
PRINT 1
PRINT1
```

U posljednjem primjeru vidite da je moguće potpuno zbaciti razmake. To nikako nije dobro jer pravilno upisane naredbe s razmacima između njih pridonose čitavosti programa, ali je osobito za početnike izuzetno važno, jedini primjer kada s razmacima moramo za stati oprezni jest pisanje tekstova unutar navodnika. Svo ono što piše unutar navodnika «ono» će uzeti točno u onom obliku kako je napisano.

KOMBINACIJE IZRAZA U PRINT-NAREDBAMA

Upišite ovaj primjer.

```
P1=3.141
R=5
P=R*R*P1
A$="POVRŠINA JE "
PRINT A$
PRINT P
```

Odobreni ispis će tako izgledati, a između teksta POVRŠINA JE i ispisanog rezultata nalazi se tekst naredbe PRINT P i sve je to još odvojeno PROMPT-om. Upišite sada prethodni primjer ponovno, jedino umjesto posljednje dvije PRINT-naredbe upišite naredbu.

```
PRINT A$;P
```

Sada je ispis mnogo lepši. Tekst POVRŠINA JE i rezultat ispisa su u jednoj liniji. Vidimo da unutar jedne PRINT naredbe može biti više izraza koji su međusobno odjeljeni točkom zarezom. Točka zarez nalog je mikroračunala da sva izraza ispiše u istom redu i da između njih ostavi mali ili nikakav razmak.

Mali razmak ostavićete kada ispisujete dva broja jedan iza drugog (tada je razmak dva prazna mjesta), kada ispisujete tekst za brojeve ili broj iza teksta (tada je razmak jedno prazno mjesto).

Kada ispisujete dva teksta jedan iza drugoga, neće ostaviti nikakav razmak. To nam omogućuje da tekstove nastavljamo jedan za drugim bez potrebe da ih zbrajamo.

Unutar jedne PRINT-naredbe može se nalaziti po veliki mnogo izraza koji su međusobno odijeljeni točkom zarezom. Jedno ograničenje je da ukupna dužina nije ne može biti veća od 72 znaka.

Osim točka zareza za odjeljivanje tekstova možemo uzeti zarez. Kada "orać" naiđe na zarez, on će ispisivanje nastaviti od 15 znaka u redu, a ako ga je prošao od 30 znaka u redu, a ako je i njega prošao, prići će u nov red, i tako dalje. Pozicije na koje će se ispis zovu se tab-pozicije (zato što nam služe prilikom tablica). Možemo reći da namaskom na zarez, "orać" skaće na prvu slobodnu tab poziciju.

Ako između dva izraza napisamo dva ili više zarezova, ispis će se pomaknuti za onoliko tab pozicija koliko smo zareza upisali. Ako izraze odvajamo točkom zarezom, onda će jedna točka zarez imati potpuno isti efekt kao i više njih.

Jedno PRINT-naredbu za odjeljivanje izraza možemo upotrebljavati i točkom zarezom i zareze, već prema tome što nam na tom mjestu odgovara.

MOJ PRVI PROGRAM

Sve što smo do sada unosili u računalo izvršavalo se odmah nakon što smo ukucali. Napišite sada naredbu

```
10 PRINT 1
```

Kao što vidite, računalo nije izvršilo naredbu PRINT, a nije ni prijavilo grešku. Ako sada upišete

```
LIST
```

računalo će ponovno ispisati: `10 PRINT 1` (tako vam ispiše još nešto prišliše tipku na početku računanja, ponovno pozovite BASIC i unesite naredbu `10 PRINT 1`).

Vidimo da je računalo zapamtio liniju `10 PRINT 1`. Takva linija naziva se programskom linijom. Za razliku od komandne, nije počela brojem, ne izvršava se sve dok to ne naredimo. Pomoću ovakvih programskih linija

Takve su greške mnogo teže za spravljanje, pa zato i opasnije.

Upišite sada u program koji imate u memoriji naredbu.

```
5 PRINT "RAČUNANJE POVRŠINE KRUGA"
```

Ako sada pokrenemo program, dobit ćemo tekst

```
RAČUNANJE POVRŠINE KRUGA
POLUMJER JE 5
POVRŠINA JE 78.525
```

Pokrenite sada program ponovno, ali umjesto naredbe RUN napisite RUN 10. Vidimo da sada tekst RAČUNANJE POVRŠINE KRUGA nije ispisan. To je zbog toga što je naredba RUN 10 naredila računaku da izvršavanje programa započne od programske linije broj 10. Tako možemo izvršavanje programa započeti od bilo koje linije. Napišite sada tekst.

```
A=15
```

Znamo da smo tim varijabli A pridružili vrijednost 15. To možemo ispitati ako napišemo PRINT A. Pokrenite sada program sa RUN nakon što se završi napišite ponovno PRINT A. Vidimo da smo sada dobili vrijed-

nost 0, odnosno da je računalo zaboravilo vrijednost varijable A (rekli smo da svaka nedefinirana varijabla ima vrijednost 0). To se dogodilo zbog upotrebe naredbe RUN. Ova naredba prije nego pokrene program izbriše sve vrijednosti svih varijabli, uključujući i string-varijable.

Pokrenite sada program naredbom RUN 20. Vidimo da su sada polumjer i površina jednaki 0. Zasto se to dogodilo?

Pogledajmo sve korak po korak. Naredba RUN 20 prvo je obrisala sve varijable uključujući i varijablu R u kojoj se nalazi vrijednost polumjera. Nakon toga izvršavanje programa počelo je od linije 20. U toj liniji definirana je vrijednost varijable PI, zatim je u naredbi 30 izračunana vrijednost varijable P, a pri tome je upotrebljena varijabla R. Kako smo liniju 10 u kojoj se pridružuje vrijednost varijabli R preskočili, u trenutku izračunavanja površine varijabla R imala je vrijednost nula. Vidimo da se prilikom pozivanja programa iz sredine mogu desiti greške, pa zato s takvim povezivanjem treba biti oprezan.

Napišite sada RUN 11. Računalo će napisati

```
?NN GOTO 5 A
```

NH je kralica od nema naredbe, odnosno »orao« nas obavještava da smo zahtijevali izvršenje linije koja ne postoji.

Slično kao i u naredbi RUN i za naredbe LIST može se nalaziti broj. Napišite:

LIST 30

Na ekranu vidite ispisanu liniju 30, odnosno naredba LIST 30 nalog je računaru da izlista liniju 30. Ako u naredbi LIST navedete broj koji ne postoji ->orao- neće ispisati ništa. Iza naredbe LIST možete navesti i dva broja odvojena crticom. Na primjer, naredba LIST 20-35 ispisat će sve linije od 20 do 35 uključujući i linije 20 i 35. Ako izostavite zadnji broj bit će izlistane sve linije od linije koju ste naveli do kraja, a ako izostavite prvi broj bit će izlistane sve linije od početka do linije koju ste naveli.

Probajte:

LIST 10-

LIST -30

Ove mogućnosti najviše će vam konstiti kada budete izlistavali duge programe koji ne stanu odjednom na ekran. Tada će nakon naredbe LIST bez navedenih granica cijeli program -preletjeti- ispred vaših očiju i vidjet ćete samo posljednjih tridesetak linija.

ISPRAVLJANJE TEKSTA PROGRAMA (EDITOR)

Ponekad je potrebno promijeniti nešto u već napisanom programu. To činimo bilo zbog toga što imamo grešku u programu, ili zbog toga što želimo promijeniti neke vrijednosti u programu. Do sada smo se služili mogućnošću da umjesto linije s greškom upišemo ispravnu liniju. <Orao> nam pruža mogućnost uređivanja programa na ekranu. To se na engleskom jeziku naziva screen editing, pa zato kažemo da <orao> ima skriven editor.

Prilikom ovog posla služimo se tipkama za pomicanje kursora (to su one četiri tipke sa strelicama) i tipkom copy (to je tipka sa oznakom PF4). Pogledajmo to na primjeru. Ukucajte doslovno ovaj program:


```
10 PRINT "ZDRAVO ";
20 GOTO 10
```

Pokrenite ovaj program sa RUN. «Orao» će početi ispisivati ZDRAVO i kad ispuní cijeli ekran pomicat će tekst prema gore i doje dodavati novu liniju spunjenu tekstom. Ovaj program izvršavać će se sve dok ga ne prekinemo. Prekid programa postićemo pritiskom na tipke (CTL) C (morate ih zadržati nekoliko sekundi). Kad ih pustite na ekranu će se pojaviti bijeli kvadratić. Nakon što pritisnemo (CR) «orao» će napisati:

```
STOP U 10 i 11
STOP U 20
```

što ovisi o tome koju je liniju izvršavao u trenutku kada smo ga prekinuli. Proanalizirajmo malo ovaj program.

Linija 10 nalog je računalu da ispiše riječ ZDRAVO. Točka zarez na kraju ove naredbe nalog je da se slijedeći tekst piše u istom redu s malim razmakom. Nakon što izvrši ovu liniju, računalo naide na liniju 20 u kojoj stoji naredba GOTO. Go to na engleskom znači = idi na. Broj iz naredbe GOTO govori računalu od koje naredba će se nastaviti izvršavanje programa. U našem primjeru to je linija 10 pa računalo ponovno izvrši liniju 10. Nakon nje ponovno naide na liniju 20 i sve se opet ponavlja. Kažemo da računalo izvršava petlju

S obzirom na to da se ova petlja izvršava sve dok je ne prekinemo, nazivamo je beskonačnom petljom. Pogledajte ovaj program:

```
10 A=1
20 A=A+1
30 PRINT A
40 GOTO 20
```

Pokrenite program i vidjet ćete da ispisuje rastući niz brojeva. Proanalizirajmo zajedno taj program.

Linija 10 postavlja u varijablu A vrijednost 1. Linija 20 uzima vrijednost varijable A, dodaje joj 1 i rezultat smjesti nazad u varijablu A. (Ovdje vidimo da znak jednakosti ima funkciju pridruživanja jer izraz $A = A + 1$ nije matematički prihvatljiv). Ovu naredbu nazivamo još i brojac jer ju često upotrebljavamo u programima kada nam je potrebno brojenje. Linija 30 ispisuje vrijednost varijable A, a linija 40 vraća izvođenje programa na liniju 20 koja ponovno povećava vrijednost varijable A. Ponovno imamo beskonačnu petlju ali se ovaj puta unutar nje mijenja vrijednost varijable A. Dakako i ovaj ćemo program prekinuti uz pomoć tipki (CTL) C.

Naredbu GOTO možemo iskoristiti i za promjenu toka programa bez zatvaranja petlje. Proanalizirajte sami zašto se u slijedećem primjeru ne ispisuje broj 2

```

10 PRINT 1
20 GOTO 40
30 PRINT 2
40 PRINT 3

```

Osim beskonačnih petlji postoje i petlje sa određenim brojem ponavljanja. Počnimo s jednostavnim primjerom.

```

10 FOR A=1 TO 10
20 PRINT A
30 NEXT A

```

Pokrenite ovaj program i vidjet ćete da ispisuje brojeve od 1 do 10. Ovdje se srećemo sa jednim parom naredbi koje uvijek idu zajedno. Prva naredba je naredba FOR unutar koje se nalazi jedna varijabla. Naredba FOR u liniji 10 znači – mijenja A od 1 do 10. Nakon toga na ovu naredbu računalo pridruži varijablu A vrijednost 1 i nastavi izvođenje. Nakon što ispiše varijablu A u liniji 20, računalo nađe na liniji 30 u kojoj piše NEXT A. Ova naredba znači – sljedeći A. Nakon nje računalo će se vratiti na naredbu 10, pridruži varijablu A vrijednost 2. Nakon 10 krugova varijabla A imaće vrijednost 10. Kada računalo nađe na naredbi NEXT A, skočit će na naredbu 10, povećati A za jedan i tada ustanoviti da je A sada veći od gornje granice navedene u naredbi FOR. Zato će se izvršavanje programa nastaviti

iza naredbe NEXT. Kako mi iza ove naredbe nemamo drugu naredbu, izvođenje programa je gotovo.

Takve petlje se unaprijed određenim brojem ponavljanja upotrebljavamo mnogo češće od beskonačnih petlji. U primjeru vidimo program koji ispisuje kvadrate brojeva od 5 do 15:

```

10 D=5
20 G=15
30 FOR A=D TO G
40 PRINT "A= ";A,"A*A= ";A*A
50 NEXT A
60 PRINT "KRAJ"

```

U liniji 30 vidimo da za određivanje granica možemo uzeti i varijable a ne samo konstante. Isto tako treba uočiti da se nakon što se dosegne gornja granica izvršavanje programa nastavlja na liniju ispod linije koja sadržava NEXT A, pa se zbog toga na kraju ispisuje riječ KRAJ.

Ponekad nam je potrebno da se varijabla u petlji mijenja s korakom različitim od 1. Tu ćemo upotrebiti naredbu STEP koja je dio naredbe FOR, što se vidi iz primjera.

```

10 FOR A=1 TO 100 STEP 7
20 PRINT A
30 NEXT A

```

Vidimo da se u ovom primjeru izvršava pet koraka. Isto tako vidimo da gornja granica (100) nije postignuta. U ovom FOR nije dovoljno sigurnost varijable prema gornju granicu. Završavanje petlje se prekida.

U ovom slučaju, kada bi se postigla granica je potrebno da je donja granica veća od gornje. To vidimo u primjeru.

```
10 FOR A=5 TO 1 STEP -1
20 PRINT A
30 NEXT A
```

Vidimo da kod izvršava pet koraka (pet koraka manje ako je step bio negativan).

U ovom slučaju da se ja koristi u primjeru petlje FOR, imamo, tako da kod izvršava pet koraka, ali se petlja prekida prije nego što se postigne donja granica.

U ovom slučaju, kada se koristi STEP u FOR, petlja izvršava A naredbu, dok STEP izvršava naredbu STEP 1.

Također, preporučljivo je vidjeti kako se izvršava petlja u ovom slučaju. I u ovom slučaju, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica.

Vidimo da kod izvršava pet koraka. Ovo nije praktično jer ako naredbu koja izvršava petlju, treba izvršiti više puta, onda se petlja prekida prije nego što se postigne donja granica. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica.

U ovom slučaju, kada se koristi STEP u FOR, petlja izvršava A naredbu, dok STEP izvršava naredbu STEP 1. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica.

U ovom slučaju, kada se koristi STEP u FOR, petlja izvršava A naredbu, dok STEP izvršava naredbu STEP 1. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica.

```
10 FOR A=1 TO 10
20 FOR B=A TO A+3
30 PRINT B;
40 NEXT B
50 PRINT
60 NEXT A
```

U ovom primjeru, kada se izvršava 30 PRINT, petlja izvršava naredbu, dok STEP izvršava naredbu STEP 1. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica.

Kada izvršava petlja, petlja se prekida prije nego što se postigne donja granica. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica. Također, kada se izvršava petlja, petlja se prekida prije nego što se postigne donja granica.

da varijabla A ima vrijednost 3. Tu vrijednost pridružili smo varijabli A kada je računalo izvršavalo naredbu INPUT u liniji 10. Dakle, naredba INPUT zaustavlja izvršenje programa i zahtijeva od nas da unesemo podatak koji se pridružuje varijabli navedenoj u naredbi INPUT.

Pokrenite ponovno prethodni program. Kada računalo napisaše upitnik, otipkajte neko slovo umjesto broja. Računalo neće razumjeti to što smo mu napisali jer očekuje brojnu vrijednost. Zbog toga će napisati

7 PONOVI UPIS

Ti me nam računalo javlja da nije razumjelo to što smo upisali i zahtijeva od nas da upišemo pravilnu vrijednost koja će se onda pridružiti varijabli navedenoj u naredbi INPUT. Vidimo da prilikom izvršenja naredbe INPUT nije dopušteno unijeti varijablu ili izraz umjesto brojne vrijednosti.

Napisać sada INPUT A kao komandu bez broja naredbi. Vidite da je računalo prijavilo grešku. Naredba INPUT može se naznačiti eksplicitno u BASIC liniji, nije ju dozvoljeno navoditi kao komandu.

U našem primjeru, mi smo možemo dvije varijable i za svaku od njih koristiti smo posebnu naredbu INPUT. Nas prethodni program možemo napisati ovako:

```
10 INPUT A,B
20 P=A*B
30 PRINT "POVRŠINA JE ";P
40 GOTO 10
```

Kada sad pokrenemo program računalo će ispisati upitnik ali računalo sada očekuje da mu damo dva broja. Možemo zbog toga napisati 3.4 (rekli smo da decimalne brojeve pišemo s točkom, na primjer 3.4, a da zarez služi za odvajanje brojeva). Kada upišemo 3.4 računalo će uzeti prv. broj (broj 3) pridružiti ga prvoj varijabli navedenoj u naredbi INPUT (varijabli A). Nakon toga će uzeti drugi broj, pridružiti ga drugoj varijabli navedenoj u naredbi INPUT (varijabli B). Poslije toga program će se normalno izvršavati kao i u prvom primjeru.

Ako u ovom programu, kada se pojavi upitnik, upišemo samo jedan broj, računalo će uzeti taj prv. broj pridružiti ga varijabli A, a zatim će napisati u sijedecem redu dva znaka upitnika (??). Ti me nas obavještava da nismo unijeli dovoljno brojeva. Možemo unijeti drugi broj i program će se normalno dalje izvršavati.

Unesemo li kada računalo očekuje dva broja više brojeva (na primjer 3,4,5,6) računalo će ispisati

7 PREVIŠE ULAZA

Ti me nas računalo obavještava da smo unijeli previše brojeva. Varijabli A-B pridruženi su prva dva broja koja smo unijeli, a ostale brojeve je računalo ignoriralo.

Greška

7. PREVIŠE ULAZA

ne prekida izvršenje programa, za razliku od g-osaka s kojima smo se do sada sreli.

U naredbi INPLT možemo navesti i string-varijablu. Tada će računalo kada naiđe na naredbu INPUT očekivati da mu unesemo nek tekst, odnosno računalo će sve što odčitamo shvatiti kao tekst i pridružiti string-varijablu koju je navedena u naredbi INPUT. Pogledajmo to na primjeru (prvo obristite prethodni program sa NEW):

```
10 INPUT A$
20 PRINT A$
30 GOTO 10
```

U ovom programu nije moguće dobiti obavijest 7-PONOVNI UPIS zbog toga što bilo koji tekst koji unesemo može biti pridružen string-varijabli. Treba, naravno, imati na umu da ukupna dužina stringa ne smije biti veća od 255 znakova.

Kao i pri numeričkim varijablama unutar jedne INPUT naredbe možemo navesti više string-varijabli odvojenih zarezom. Kada računalo napíše upitnik za svaku od tih string-varijabli, napisati ćemo po jedan

tekst. Tekstove ćemo udijavati zarezom. Ako želimo da u tekst koji unosimo u naredbi INPUT upišemo zarez, tada tekst moramo navesti u navodnicima. Navodnici će «orao» ignorirati. Unutar iste naredbe INPUT možemo navesti i numeričke i string-varijable. Kada odgovaramo na komandu INPUT moramo tekstove i brojeve unositi istim redom kojim su spisane varijable unutar naredbe INPUT.

Upisite ovaj program

```
10 INPUT A$
20 INPUT A,B
30 S=B-A
40 PRINT A$;" IMA ";S;" GODINA"
50 GOTO 10
```

Pokrenite ovaj program i kad budete prvi upitnik upisali svoje ime P i tiskate CR na drugi upitnik napisate prvo godinu svoga rođenja a zatim «ekuću godinu» (ova u kojoj smo sada). Vidi da računalo zna ispisati koliko imate godina. To njegovo «znanje» rezultat je programa koji smo mi napisali.

Pomislili ste vjerojatno da je dosta teško zapamtiti što pojedina naredba INPUT očekuje da joj unesemo. Možemo unutar INPUT-naredbe napisati u navodnicima tekst koji će nam pomoći da pravilno odgovorimo

10 REM *** DIO JE REM NAREDBA ***

Pokrenite program s RUN. Vidite da se nije ništa dogodilo. Naredba REM služi nam isključivo za stvaranje teksta u program i nije računalo potpuno ignorira. Za vrijeme izvršenja programa. Prema tome izvršenje nekog programa neće se mijenjati ako u programu postoje naredbe REM.

U naredbi REM može se nalaziti bilo kakav tekst, a dopuštena je i upotreba svih znakova, jer računalo ovaj tekst nikada ne čita. Unutar naredbe REM nije moguće učiniti grešku.

S naredbama REM ne treba biti štedljiv i ne treba ni prelijavati. Prevelika količina naredbi REM šteti je čitljivosti programa jednako kao da ih i nema.

Tekst koji je upisan u naredbu REM ne treba biti previše dugačak. Na primjer:

```
10 REM *** DIO PROGRAMA ZA IZRAČUNAVANJE PLOŠTINE KVADRATA ***
```

To je previše dugačka naredba REM i zbog toga nećite ka. Ne valjaju ni previše kratke naredbe REM, na primjer

```
10 REM IPK
```

Tesko da će iz ovoga netko shvatiti da se izračunava površina kvadrata. Za naredbu REM je najbolje da je duga jednu liniju i da sadržava najbitnije podatke o dijelu programa koji opisuje. U danom primjeru, na bolje bi bilo napisati:

10 REM PLOŠTINA KVADRATA

Jos jednom želim naglasiti vrijednost naredbi REM zbog toga što ih mnogi programeri ne stavljaju. To dovodi do toga da imamo programe koje je teško čitati i još teže prepravljati. U naredbom primjeru u naredbi REM opisani su neki dijelovi programa

```
10 REM PLOŠTINA KVADRATA
20 INPUT "KOLIKA JE STRANICA ";
S
30 REM RAČUNANJE PLOŠTINE
40 P=S*S
50 PRINT "PLOŠTINA JE ";P
60 REM SKOK NA POČETAK
70 GOTO 10
```

DONOŠENJE ODLUKA (IF)

Pri programiranju vrlo često nailazimo na probleme o kojima računalno treba donijeti neku odluku. Na primjer, ako računamo opseg kruga nije moguće da poluprec kruga ima negativnu vrijednost (ne postoje negativne dužine). To znači da u programu moramo testirati da li je poluprijer negativan i ako jeste, prijaviti da je nastala greška. Za testiranje pojedinih brojeva služiti ćemo se dodatnim operatorima. Usporedimo li dva broja, moguća su tri različita stanja:

- prvi broj je veći od drugoga
- prvi broj je manji od drugoga
- oba broja su jednaka

Za svako od ovih tri stanja imamo odgovarajuće operatore. To su znaci veća ($>$), manje ($<$) i jednako ($=$).

Da biste lakše razlikovali znakove manje i veće, zapamtite da je vrh uvijek okrenut prema manjem broju. I nije se šire prema većem broju.

Osim ove tri osnovne usporedbe ponekad će nas zanimati neka od složenih usporedbi. Postoje tri složene usporedbe, i to su:

- prvi broj je veći ili jednak drugome
- prvi broj je manji ili jednak drugome
- prvi broj je različit od drugoga

Za složene usporedbe imamo složene operatore. To su: veći-jednako ($>=$), manje-jednako ($<=$), različito (\neq).

Vratimo se našem problemu negativnog poluprijera. Da bismo ustanovili da li je nek broj negativan, poslužiti ćemo se spoznajom da su negativni brojevi manji od 0. Pogledajmo

```
10 INPUT "POLUPRIJER ";R
20 IF R<0 GOTO 10
30 PI=3.141
40 O=2*R*PI
50 PRINT "OPSEG JE ";O
60 GOTO 10
```

Naprije pokrenite ova, program i unesite neku pozitivnu vrijednost za poluprijer R. Vidjet ćete da program ispravno radi. Sada unesite negativnu vrijednost. Kao što vidite, program vas ponovno pita za vrijednost po-

Umjera, odnosno ne prihvaća negativne vrijednosti. Pogledajmo kako smo to postigli. Nakon što se u naredbi 10 unese vrijednost varijable R, računalo dolazi na liniju 20. U toj liniji nalazi se naredba IF (IF na engleskom znači ako). Iza riječi IF dolazi uvjet. Uvjet je izraz koji sadrži jedan od operatora za usporedbu. Ovdje to je test da li je R manji od 0, odnosno da li je R negativan broj. Iza uvjeta nalazi se naredba koja se izvršava SAMO AKO JE UVJET ISPUNJEN. Dakle, liniju 20 možemo pročitati ovako: ako je R manje od 0, onda idi na liniju 10.

Vidjeli smo da računalo pravilno izračunava vrijednost ako je R pozitivan broj. Naredba IF ne izvršava tekst iza uvjeta, ako uvjet nije zadovoljen. Jednostavno je rećeno, ako uvjet u naredbi IF nije zadovoljen, izvršavanje programa nastavit će se na prvu naredbu ispod naredbe IF.

Kao što smo vidjeli, naredba IF može izazvati uvjetovani skok u programu. Takav uvjetovani skok vrlo često je primjenjivan u svim ozbiljnim programima. U slijedećem primjeru iskoristit ćemo naredbu IF da odaberemo dio programa koji će se izvršiti. Prvo ćemo unijeti dva broja, a zatim znak plus ili minus. Ako unesemo plus, piva dva broja će se zbrojiti, ako unesemo minus, oduzet će se drugi broj od prvoga. «O-ao» neće prihvatiti nista drugo osim plusa i minusa.

```
10 INPUT "PRVI BROJ ";A
20 INPUT "DRUGI BROJ ";B
```

```
30 INPUT "+ ILI - ";AS
40 IF AS="+" GOTO 100
50 IF AS="-" GOTO 200
60 GOTO 30
100 REM ZBRAJANJE
110 S=A+B
120 PRINT "ZBROJ JE ";S
130 GOTO 10
200 REM ODUZIMANJE
210 S=A-B
220 PRINT "RAZLIKA JE ";S
230 GOTO 10
```

U liniji 40 vidimo da je u uvjetu unutar naredbe IF moguće usporediti i stringove. U tome više u pojavi koju je govorio o tome kako «orać» pamti slova.

Vidimo da u programima 40 i 50 testiramo da li je u string-varijabli AS plus ili minus. Pazite: ako nije zadovoljen uvjet ni u naredbi 40 računalo odlazi u naredbu 50, a ako nije zadovoljen uvjet ni u naredbi 50 izvršava se naredba 60, odnosno skače se na liniju 30, ponovno postavljajući pitanje o matematičkoj operaciji. Vidimo da će linija 60 biti izvršena jedino ako nisu zadovoljeni uvjeti ni u naredbi 40, ni u naredbi 50. Takve točke u kojima se tok programa mijenja pod nekim uvjetom, nazivamo još i grananje programa.

Iza uvjeta u naredbi IF ne mora biti naredba GOTO.

Ako želimo staviti neku drugu naredbu, tada moramo iza uvjeta napisati riječ **THEN** za te riječi možemo napisati bilo koju drugu BASIC-naredbu, ona će se izvršiti jedino ako je uvjet ispunjen. Iza riječi **THEN** može stajati i druga naredba **IF**, međutim time program postaje znatno zamršeniji, teži za pracenje, pa se to vrlo rijetko primjenjuje. Pripravili ćemo nas prethodni primjer uz upotrebu riječi **THEN** u naredbi **IF**.

```

10 INPUT "PRVI BROJ ";A
20 INPUT "DRUGI BROJ ";B
30 INPUT " + ILI - ";A$
40 IF A$="+" THEN PRINT "ZBROJ
JE ";A+B
50 IF A$="-" THEN PRINT "RAZLIK
A JE ";A-B
60 GOTO 30

```

Vidimo da je ovako napisan program kraći od prethodnog primjera. Prema tome, oblik **IF ... THEN** upotrijebit ćemo kad god je to moguće jer time program dobiva na čitkosti. To je zbog toga što nije potrebno skakati tok programa kroz bespotrebne skokove. Odmah vidimo što će se izvršiti ako je uvjet ispunjen, za riječi **THEN** može se nalaziti naredba **GOTO**, međutim tada **THEN** nije potreban pa ga možemo zostaviti. U sljedećem primjeru imamo program koji ispisuje brojeve od 1 do 10 ili ne ispisuje broj 7.

```

10 FOR A=1 TO 10
20 IF A=7 THEN A=A+1
30 PRINT A
40 NEXT A

```

Vidimo da naredba **IF** koja se nalazi unutar petlje mijenja vrijednost kontrolne varijable **A** onda kada **A** ima vrijednost 7. Zbog toga se vrijednost 7 ne ispisuje.

Osim testiranja određenih stanja, pomoću naredbe **IF** možemo formirati petlju naizgled petlje **FOR**. Pogledajmo primjer.

```

10 A=0
20 A=A+1
30 IF A>10 GOTO 60
40 PRINT A
50 GOTO 20
60 PRINT "KRAJ"

```

Vidimo da se u naredbi 30 testira da li je **A** veći od 10; ako nije, skine na liniju 20. Tako formirana petlja u potpunosti je nalik na petlju **FOR**. To je zbog toga što se uvjet testira nakon povećavanja varijable **A** jednako kao što to čini naredba **NEXT** u petlji **FOR**. Osnovna sličnost između ovakve petlje i petlje **FOR** je u tome što će nakon završetka petlje broj biti veći od postavljene gornje granice. Pogledajmo primjer


```

10 A=0
20 IF A=10 GOTO 60
30 A=A+1
40 PRINT A
50 GOTO 20
60 PRINT "KRAJ"

```

Ovdje se spisuju potpuno isti brojevi kao i u prethodnom primjeru, jedino što je varijabla A testirana prije nego što je povećana pa će po izlasku iz petlje imati vrijednost jednaku gornjoj granici određenoj u petlji. Ova, drugi postupak primjenjujemo mnogo češće, pogotovo kada radimo prerage nekih podataka. Umjesto onog prvog postupka, uvijek je bolje raditi petlju FOR.

Pri formiranju petlje upotrebom naredbe IF treba paziti na to da ako se uvjet testira na ulazu u petlju, onda je moguće da se petlja ne izvrši niti jednom (ako je uvjet zadovoljen već pri prvom testiranju). Ako uvjet testiramo na kraju petlje, onda će se ta petlja izvršiti najmanje jednom bez obzira na to da li je pri prvom testiranju uvjet zadovoljen ili nije. To je ilustrirano u narednja dva primjera.

```

50 GOTO 20
60 PRINT "KRAJ"

```

```

10 REM TEST NA KRAJU
20 A=0
30 PRINT A
40 IF A=0 GOTO 60
50 GOTO 20
60 PRINT "KRAJ"

```

U prvom primjeru linije 40 i 50 uopće se ne izvršavaju, a u drugom primjeru izvršava se linija 30 bez obzira na to što je već u prvom proazvu zadovoljen uvjet. Dakle, sve linije prije testa s kojim izlazimo iz petlje izvršavaju se jedan puta više nego linije iza testa.

```

10 REM TEST NA ULAZU
20 A=0
30 IF A=0 GOTO 60
40 PRINT A

```

NAREDBE ZA PREKIDANJE PROGRAMA

Povremeno se javi potreba da izvođenje nekog programa prekinemo kada je u programu nastupilo određeno stanje. To postizemo naredbom STOP čije značenje nije potrebno objašnjavati. Ova naredba može se nalaziti na kraju nekog dijela programa koji se izvršava samo onda kada je počinjena greška ili unutar IF naredbe da bi se prekinulo izvođenje programa na zahtjev korisnika. Naredni program ilustrira upotrebu ove naredbe.

```
10 INPUT "UNESI DVA BROJA ";A,B
20 PRINT "NJIHOV ZBROJ JE ";A+B
30 IF A=0 THEN STOP
40 GOTO 10
```

Ovaj program zbraja dva unesena broja i radi to sve dok prvi uneseni broj nije 0. Kada pretpostavi da je prvi uneseni broj 0, program se prekida uz pomoć naredbe STOP. «Orao» će ispisati ovaj tekst:

STOP U 30

Time nas obavještava da je prekid nastupio zbog naredbe STOP a ispisuje nam liniju u kojoj je našao tu naredbu. Nakon što je program prekinut naredbom STOP, mi smo ga prekinuli pritiskom na (CTL) C, možemo nastaviti izvođenje programa ako otkucamo

CONT

CONT je kratica od continue što na engleskom znači nastaviti.

Naredba STOP ima specifičnu primjenu prilikom testiranja programa. Ako u programu imamo više grananja i pojavi li se nam se jedna greška u nekom od tih grananja, a ne znamo u kojem upotrijebit ćemo nekoliko naredbi STOP. U svaku granu programa ubaciti ćemo STOP naredbu i zatim pokrenuti program. Kada program nađe na naredbi STOP, «orao» će ispisati broj linije, odnosno na koju je STOP naredbu naišao. Sada ćemo pogledati da li se sve zbiva prema planu, ako je sve dobro, program ćemo nastaviti sa CONT. Ovaj postupak ponavljat ćemo sve dok ne otkrijemo grananje u kojem je greška.

BASIC i kraćunala «orao» posjeduje i naredbu END. Ova naredba zaostala je iz vremena velikih računala koja su imala mnogo terminala, na se u svakom BASIC-programu morao označiti kraj programa. Ovu naredbu možemo staviti na kraj našeg programa, to neće izmijeniti njegovo izvršavanje. Naredbu END mo-

žemo i pametnije upotrijebiti. Ako želimo program završiti usred neke petlje, bez skakanja na njegov kraj možemo na tom mjestu napisati END. Kada «dora» nađe na naredbu END, on će postupiti jednako kao i kad izvrši posljednju naredbu u programu.

Program koji smo završili naredbom END ne možemo nastaviti naredbom CONT.

Funkcija INT služi nam za dobivanje cijelog broja zaokruživanjem na PRVI MANJI BROJ. To znači da će nam funkcija INT u pozitivnim brojevima davati za rezultat cjelobrojni dio broja, odnosno ono što se nalazi ispred decimalne točke. Na primjer, ako zatražimo

PRINT INT (3.5)

racunalo će ispisati broj 3. Vidimo da je broj 3 prvi manji cijeli broj ispred broja 3.5.

S negativnim brojevima sve je nešto kompliciranije. U primjeru

PRINT INT (-3.5)

dobit ćemo rezultat -4. Ako se sjetimo skupa cijelih brojeva, vidjet ćemo da je broj -4 prvi manji cijeli broj ispod broja -3.5. To treba imati na umu kada se radi s funkcijom INT, jer možemo pogriješiti u programu.

Ako želimo pravilno zaokruživanje broja uz funkciju INT, moramo prvo na broj pribrojiti vrijednost 0.5. To slijedi iz pravila o zaokruživanju brojeva. Svi brojevi čiji decimalni dio ima vrijednost manju od 0.5 zaokružuju se nadolje, a svi oni kojima je decimalni dio veći ili jednak 0.5 zaokružuju se nagore. Kada na nek. broj pribrojimo 0.5 a njegov decimalni dio je bio manji od 0.5, cjelobrojni dio broja neće se promijeniti. Ako 0.5 pribrojimo na broj čiji je decimalni dio veći ili jednak 0.5, cjelobrojni dio broja povećat će se za jedan. U oba slučaja funkcija INT dat će nam nakon pribrajanja 0.5

FUNKCIJA INT

Ovo je prva iz skupine funkcija o kojima će kasnije biti mnogo više govora. Za početak recimo samo da je svim funkcijama zajedničko to što se iza njih navodi jedan podatak u zagradili to se naziva argument funkcije i one daju određeni rezultat koj možemo spisati ili pridružiti varijabli. Zbog toga funkcije ne možemo navoditi samostalno u linijama BASIC programa, već moraju biti unutar nekog izraza.

željeni rezultat. To možete vidjeti u narednom programu u koj možete unositi broj koji decimalni broj, i program će ga pravilno zaokružiti.

```
10 INPUT A
20 PRINT INT (A+.5)
30 GOTO 10
```

U ovom primjeru vidimo također da je broj, 0.5 u liniji 20 napisan kao .5 što mikroračunalo dopušta. Možda ste primijetili da ispisuje sve brojeve koji su manji od 1 ispisuje bez nule ispred decimalne točke.

Osim zaokruživanja na cijele brojeve ponekad nam je potrebno zaokruživanje na određen broj decimalnih mjesta. U primjeru zaokruživanja na dvije decimale objasniti ćemo kako se to radi. Ako želite zaokruživanje na tri decimale upotrijebit ćete umjesto broja 100 broj 1000, za četiri decimale broj 10 000 itd.

```
10 INPUT A
20 A=A*100
30 A=A+.5
40 A=INT(A)
50 A=A/100
60 PRINT A
70 GOTO 10
```

Proučimo ovaj primjer detaljno. Nakon što smo u liniji 10 unijeli neki broj (na primjer 6.6666), u liniji 20 pomnožili smo taj broj sa 100 (666.66). Na ovako dobiveni broj primijenili smo već prije opisani postupak zaokruživanja. Dakle, prvo smo pribrojili .5 (667.16), a zatim potražili INT od tog broja (667). Na kraju smo u liniji 50 podijelili taj broj sa 100 (6.67). Vidimo da smo na kraju dobili broj spravno zaokružen na dvije decimale.

Funkciju INT možemo korisno upotrijebiti i onda kada želimo ispitati da li je neki broj cijeli broj. U narednom primjeru ispred cijelih brojeva štampane su zvjezdice. To je postignuto u naredu 20 gdje se kontrolira da li je broj A jednak INT(A). Ako jeste, u pitanju je cijeli broj.

```
10 FOR A=0 TO 10 STEP .5
20 IF A=INT(A) THEN PRINT "*";
30 PRINT A
40 NEXT A
```

Osim za provjeru da li je u pitanju cijeli broj, funkciju INT možemo upotrijebiti i za to da ispitamo da li je neki broj djeljiv s nekim drugim brojem. Sve se izvodi slično odnosno prvo ćemo podijeliti broj koji ispitujeemo, a zatim ćemo provjeriti da li smo dobili cijeli brojni rezultat. U narednom primjeru program ispisuje brojeve od 1 do 50 ali ne ispisuje brojeve koji su djeljivi sa četiri.


```

10 FOR A=1 TO 50
20 B=A/4
30 IF INT(B)-B THEN GOTO 50
40 PRINT A
50 NEXT A

```

programa slično. Naravno, slučajni brojevi imaju široku primjenu u igrama. Unesite ovaj program:

```

10 FOR A=1 TO 10
20 PRINT RND (7)
30 NEXT A

```

Vidimo na ekranu ispis deset različitih brojeva. Pogledajmo prvo i niju 20 koja je ispisivala te brojeve. U njoj naznačimo funkciju RND i za nje broj sedam u zagradu. Ovaj broj potrebno je napisati ako njegova vrijednost nije bitna. To može bit bilo koji pozitivan broj različit od nule. Ovaj broj "odlu" služi prilikom zračunavanja slučajnog broja, ali to ne znači da će isti broj dati uvijek isti rezultat. Jedino nula daje će stalno isti rezultat, odnosno ponavljat će posljednji generiran slučajni broj.

Pogledajmo sada brojeve koje nam je program ispisao. Vidimo da su svi brojevi manji od jedan. Funkcija RND generira nam slučajni broj u rasponu od nule do jedan, koji može bit nula, a ne može biti jedan. Matematičari bi to napisali ovako:

$$0 \leq \text{RND} < 1$$

Uz pomoć jednostavne matematičke operacije i funkcije INT, možemo dobiti broj u nekom drugom raspo-

SLUČAJNI BROJ (RND)

Ovo je također riječ koja pripada u kategoriju funkcija. Pozabavit ćemo se tom funkcijom koja omogućuje mikroracunalu da «zamisli» neki slučajni broj. Da budemo pošten, moramo reci da računalo ne može izmisliti broj. Zapravo «orač» izračunava jedan broj uzimajući pri tome početnu vrijednost koja nam nije poznata i primjenjujući toliko složen obrazac da se nama čini kao da se brojevi slučajno pojavljuju. Ova funkcija, odnosno slučajni brojevi nužni su za istraživanje dinamičkih procesa, testiranje matematičkih i statističkih

nu. U slijedećem primjeru generirat ćemo brojeve u rasponu od 0 do 5.

```
10 FOR A=1 TO 10
20 B=6*RND (7)
30 PRINT INT(B)
40 NEXT A
```

Vidimo da nam ovaj program ispisuje brojeve u rasponu od 0 do 5. To smo postigli tako da smo slučajni broj u rasponu od 0 do 1 pomnožili sa brojem koji je ZA JEDAN VEĆI OD GORNJE GRANICE INTERVALA. Taj broj je za jedan veći zbog toga što je funkcija INT kao što smo rekli, zaokruživanje na prvi manji cijeli broj. Prema tome kada god množimo slučajni broj s nekim drugim brojem, moramo imati u vidu učinak funkcije INT.

Ako nam treba broj u nekom drugom rasponu (ne od nula) tada ćemo slučajni broj pomnožiti s razlikom između donje i gornje granice intervala uvećanom za jedan. Nakon toga pribrojiti ćemo donju granicu i onda uzeti INT od dobivenog rezultata. Ako je donja granica A a gornja granica B to možemo napisati ovako:

```
INT ((B-A+1)*RND(7)+A)
```

Pogledajte to na primjeru: treba nam slučajni broj u rasponu od 3 do 9. Oduzeti ćemo broj 3 od broja 9

Na dobiveni rezultat (6) pribrojiti ćemo 1 i dobiti broj 7. Ovaj broj pomnožiti ćemo sa RND i na kraju pribrojiti donju granicu (3). Tako dobiveni rezultat pretvoriti ćemo u cijeli broj pomoću funkcije INT. To je ilustrirano primjerom:

```
10 FOR A=1 TO 10
20 S=INT((9-3+1)*RND(1)+3)
30 PRINT S
40 NEXT A
```

U liniji 20 možemo jednostavnije napisati INT (7*RND (1)+3)

Na česta će nam biti potrebni brojevi u intervalu od 1 do nekog broja. Tada se sve pojednostavljuje jer nam je dovoljno da prije nego što potražimo cjelobrojni dio RND pomnožimo s gornjom granicom i dodamo 1. Slijedeći primjer ispisuje brojeve u rasponu od 1 do 100.

```
10 FOR A=1 TO 20
20 S=INT(100*RND(1)+1)
30 PRINT S
40 NEXT S
```

OPIS I PRAVILA IGRE:

Cilj je igre da pogodimo broj koji je "zamislilo" računalo. Taj broj nalazi se u rasponu od 1 do 100, a može biti i 1 i 100. Na svaki naš pokušaj da pogodimo broj računalo će ispisati da li je broj koji treba pogoditi veći ili manji od broja koji smo sami unijeli, a ako smo pogodili ispisat će nam broj pokušaja koji nam je bio potreban da dođemo do točnog rješenja.

Nakon što smo proučili pravila igre, možemo planirati program. Zadatak ćemo razbiti na manje cjeline, nakon toga svaku cjelinu obraditi posebno.

PISANJE PROGRAMA

Ako ste pažljivo pratili sve što je do sada objašnjeno, stekli ste dovoljno znanja da možete samostalno pisati programe. Dosadašnji dio knjige bavio se u prvom redu pojedinim BASIC instr. kojima. Sada treba progovoriti nešto o tome kako se programi pišu, kako se za to, posao priprema i kako se program dokumentiraju.

Da bismo to ilustrirali, proći ćemo zajedno čitav posao pisanja nekog programa. Uzet ćemo si u zadatak da napisemo kratku BASIC-ovu. Posao započinjemo tako da razmotrimo sve što znamo o samom zadatku. U igri, to je opis igre i pravila igranja.

DIJELOVI PROGRAMA:

A generiranje slučajnog broja u rasponu od 1 do 100 i postavljanje broja pokušaja na 0

B – unošenje broja i uspoređivanje sa zadanim brojem

C – obavješćavanje o rezultatu pokušaja

D – ispisivanje broja pokušaja

E – završni dijelovi igre

Ove dijelove razraditi ćemo u još manje skupine. Svaku od ovako nastalih cjelina posla ćemo prevesti u BASIC-naredbe. Takav prikaz programa nazivamo skicom programa.

SKICA PROGRAMA:

A.1 – generiranje slučajnog broja

A.2 – postavljanje broja pokušaja na 0

A.3 – ispisivanje uvodne poruke

B.1 – unošenje broja

B.2 – povećanje broja pokušaja

B.3 – ispitivanje da li je broj jednak zadanom, ako je, idi na D.1

B.4 – ispitivanje da li je broj manji od zadanog, ako je, idi na C.1

B.5 – idi na C.3 (ovdje smo sigurni da je broj veći od zadanog jer nije ni manji, ni jednak)

C.1 – ispisivanje poruke da je broj manji od zadanog broja

C.2 – idi na B.1

C.3 – ispisivanje poruke da je broj veći od zadanog broja

C.4 – idi na B.1

D.1 – ispisivanje poruke da je broj uspješno pogodan

D.2 – ispisivanje broja potrebnih pokušaja

E.1 – ispisivanje poruke da li se želi igrati još jednom

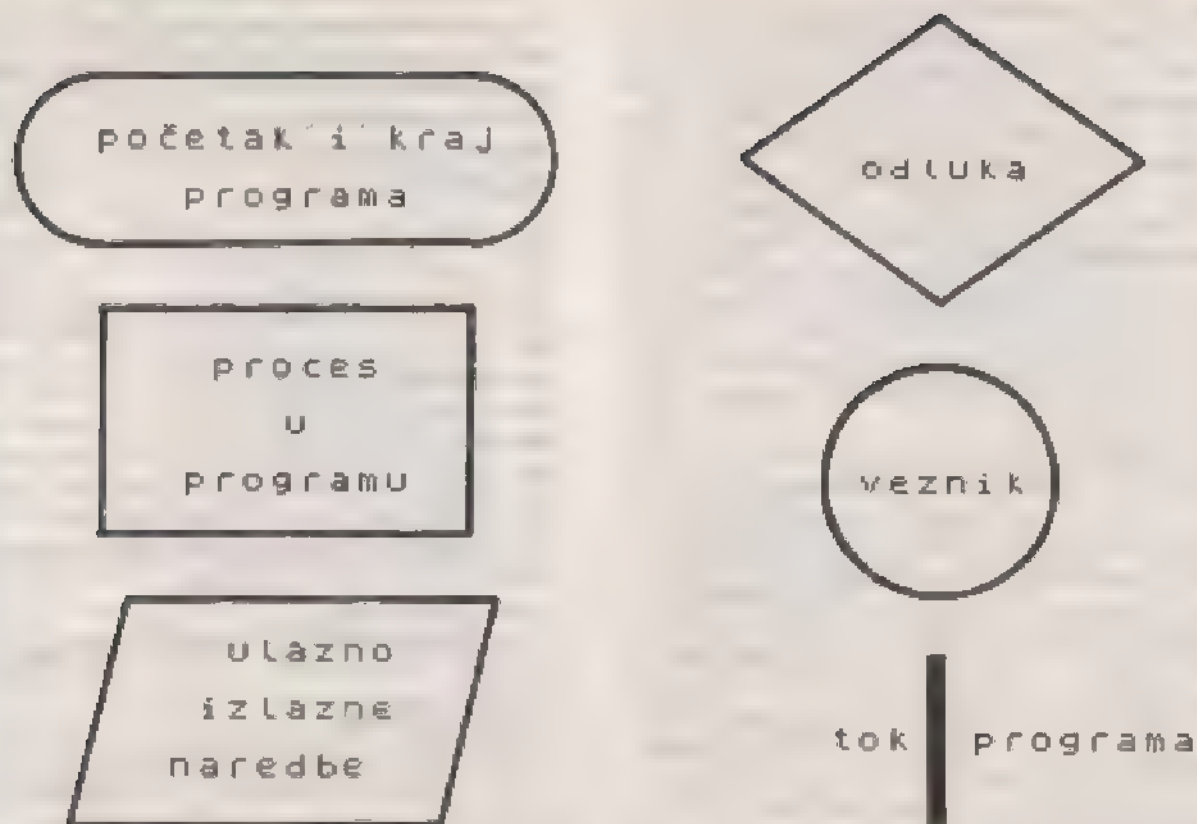
E.2 – unos odgovora

E.3 – ispitivanje da li je odgovor potvrđan, ako je, idi na A.1

E.4 – kraj programa

Skicu programa možemo prikazati grafički. To je svakako najbolji put da se prikaže i točno izradi skica programa. Tako izradenu skicu programa nazivamo dijagram toka programa ili blok-dijagram. Pri izradi ovih dijagrama služimo se određenim grafičkim simbolima koji su prikazani na slici 2. Na slici 3 vidimo dijagram toka naše igre. Vidimo da svakom elementu skice programa odgovara jedan grafički element na dijagramu toka. Toka programa u tom dijagramu odgovaraju linije koje povezuju pojedina blokov. Na mjestima gdje donosimo određene odluke linije se granaju. Zato ove točke nazivamo grananjem programa.

Slika 2




```

160 INPUT A$
170 IF A$="DA" GOTO 10
180 PRINT "DOVIDENJA"

```

Ako budete ovaj program unositi u računalo, prepisat ćete ga iz knjige. Zbog toga ne postoji potreba da program prvo napisete na papiru. Kada budete pisati svoj program, najbolje je da skicu programa prevedete u BASIC-naredbe najprije na papiru, pa da tek onda unesete program u memoriju. To napominjem zbog toga što ćete tako izbjeći mnoge greške koje bi inače nastale pri direktnom upisivanju programa. Osim toga, u naredbama 60, 70 i 80 izvodi se skok na naredbe koje se nalaze ispod ovih naredbi. Ako program pišete direktno u računalo, dok pišete liniju 60 ne znate da u njoj treba napisati skok na liniju 130 jer je još niste ni napisali. U radu na papiru vrijednost u GOTO-naredbama upisuje se tek nakon što su napisane sve linije programa.

U ovaj program nisam izvršio niti jednu REM-naredbu. Jedino sam to zbog toga da prevodenje skice programa u tekst programa bude doslovno. Između linija 20 i 30 mogu smo dodati u naredbi REM tekst POČETAK PROGRAMIJE između linija 120 i 130 može se napisati REM USPJEŠAN POGODAK i tako dalje.

Sve što ste izradili prilikom izrade programa (opis skica, dijagram toka) zajedno s papirima na kojima ste pravili program u BASIC-naredbe jest dokumentacija

programa. Kratka dokumentacija je listing programa ako možete da vaše računa o povežete na štampač. Na takav listing treba dodati određene komentare. To treba učiniti neposredno nakon završetka rada na programu, dok se još sećamo svega što smo na programu radili. Tako kompletnu dokumentaciju treba sačuvati jer će nam ona pomoći ako se javi bilo kakva potreba da u programu nešto promijenimo. Kada su u pitanju igre, mala je vjerojatnost da ćemo u njima nešto mijenjati nakon što su jednom završene, ali se može desiti da ih prenosimo na neko drugo računalo i tada će nam ponovno biti potrebna dokumentacija. Najčešće je mnogo lakše pristupiti izradi novog programa nego modificirati program za koji ne postoji prikladna dokumentacija.

Razlika između dobrog i odličnog programera najčešće je u tome što odličan programer ima dobru dokumentaciju za svoje programe.

Mnogi će vam savjetovati da je najbolje za izradu programa da sjednete za računalo i sve radite direktno iz glave. Činjenica je da to može 1% programera, ali je isto tako činjenica da 99% programera misli da to može. Nadam se da ćete vi biti bilo u onih 1% koji to zaista mogu ili barem u onih 1% koji cijelom poslu pristupaju pametno.

Kada se program piše direktno u računalo, tada se sve otkrivane greške popravljaju sitnijim bez unošenja u bilo kakvu dokumentaciju. To rezultira programom koji je potpuno napregovan, koji u sebi sadržava iko

zna koliko grešaka koje u toku testiranja nisu otkrivene, odnosno to je najčešće vrlo loš program. Takve programe zbog njihove zapetljanosti programeri nazivaju „špageti-programi“.

Posebna stavka u radu na programu jest ispitivanje programa. To se radi onda kada je program kompletno gotov. U ispitivanju programa ima dva pristupa. Prvi je pristup da program pustimo u rad i testiramo ga u radu. To je tzv. test u praksi i on će najverovatnije otkriti bilo kakvu grešku ako ona postoji u programu. Međutim, takvo testiranje zahtijeva vrlo mnogo vremena, a ponekad mnogo novaca ili je čak opasno. Zamislite da jedan program koji kontrolira kretanje podzemnih željeznica u nekom velegradu testiraju tako da ga puste da radi, pa šta bude – bude.

Drugo testiranje je da program ispitujemo blok po blok unoseći u svak blok sve moguće ulazne podatke koji u taj blok mogu doći u toku rada programa. Kad ispitamo svaki pojedini element ispitivat ćemo kako li elementi rade u međusobnom odnosu, a tek onda na kraju i cijeli program.

Testiranje programa je vrlo važan dio rada na programu jer program postaje smisljena cjelina tek nakon što iz njega uklonimo greške. Ne treba se zanositi idejom da možemo pisati programe bez grešaka, jer praksa pokazuje da i programeri s vrlo dugim programerskim iskustvom povremeno griješe, ponekad čak i najbanalnije. Za vrijeme testiranja programa na greške ne treba gledati kao na osobnu uvredu, već svom pro-

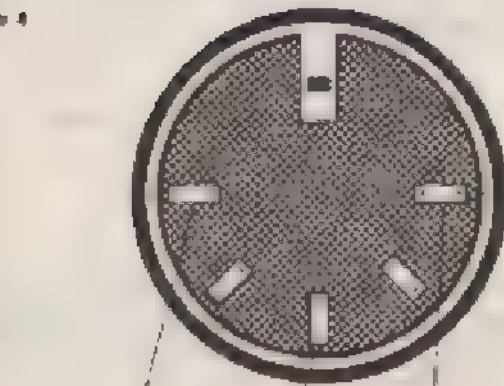
gramu treba pristupiti koliko god je to moguće objektivno i kritički. Dobro je ako testiranje programa možemo povjeriti nekom drugom jer će osoba koja nije radila program neke greške otkriti znatno lakše od autora programa.

Svaku grešku koju otkrijemo, odnosno ispravak greške, općenito svaku promjenu u programu treba odmah unijeti u programsku dokumentaciju.

ČUVANJE PROGRAMA

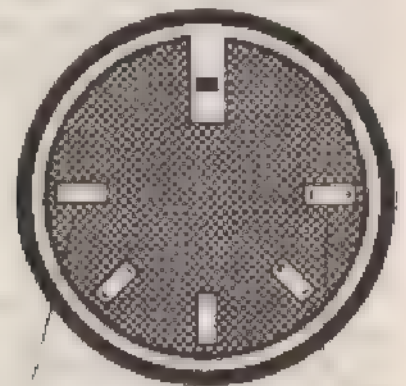
Kada bismo morali svaki puta iznova upisivati program u računalu kada nam je on potreban, ne bi bilo velike koristi od mikrorachunala. Zbog toga su konstruktori računala predvidjeli mogućnost da se programi spremaju na magnetske vrpce. Mi ćemo upotrijebiti

Slika 4



MIC masa EAR

Kasetofon



DTR masa RXD

Štampač

običan kućni kasetofon – standardne muzičke kasete. Na slici 4 vidimo kako treba povezati -orao- s kasetofonom. Nakon što smo sve povezali moramo na kasetofonu pravilno podesiti glasnoću i boju tona. Za glasnoću je najbolje da namjestimo potencijometar negdje oko sredine. Potencijometar za boju tona treba postaviti na visoke tonove. Za snimanje programa najbolje je uzeti nove vrpce. Važno je da na vrpci nema oštećenja jer program neće biti dobro spremljen.

Ako u memoriji imamo program (recimo, igru iz prethodnog poglavlja) možemo ga spremići na vrpcu. Povezat ćemo -orao- s utičnicom MIC na kasetofonu (drugi vod nećemo spojiti jer to pravi probleme u mnogim kasetofonima). Napisat ćemo:

SAVE "IGRA"

Tekst u navodnicima je ime programa – mi ga potpuno slobodno pridružujemo programu. Jedino je ograničenje da ime ne smije biti duže od 12 znakova. To ime služi nam da kasnije lakše prepoznamo program na vrpci. Nakon što smo pritisnuli (CR) -orao- će napisati.

SNIMANJE ?

Uključimo snimanje na kasetofonu i pritisnemo (CR). Iz -orao- ćemo začuti pištavi zvuk što je znak da se

program snima na vrpcu (zapravo program se snima u obliku tog tona). Kada nam se pojavi PROMPT i kursor, snimanje programa je završeno. Tako snimljeni program možemo uvijek ponovno unijeti u memoriju. To radimo ovako:

Povežemo -orao- s utičnicom EAR na kasetofonu (opet je dobro drug vod isključiti). Premotamo vrpcu na početak programa. Upišemo.

LOAD "IGRA"

Nakon što pritisnemo (CR) -orao- će ispisati

REPRODUKCIJA ?

Pritisnemo PLAY na kasetofonu (CR) na -orao-. Nakon nekog vremena na ekranu će se pojaviti PROMPT i kursor. Sada možemo sa LIST pogledati naš program i naravno, pokrenuti ga sa RUN.

Ako smo zaboravili ime programa možemo upisati samo

LOAD ""

Tada će -orao- uzeti prvi program na koji naiđe na vrpci.

Ako se program ne može učitati ili ako se učitava ali je oštećen, najvjerovatnije imamo problema s nepravilno podešenom glasnoćom i visinom tona. Pravilne vrijednosti najlakše ćemo pronaći eksperimentiranjem.

Na kasetu treba zapisati koje smo programe na nju snimili, a ako kasetofon ima brojke, možemo napisati i broj od kojeg program počinje.

Postoji naredba LOADC koja će ispisivati sve programe na koje nađe na vrpci. Ispisival će se ime programa, startna adresa, završna adresa i dužina programa. Ovi brojevi ispisani su heksadecimalno.

POTPROGRAMI

Pri pisanju dužih programa dešava se da na nekoliko mjesta ponavljamo isti skup naredbi. Na primjer, više puta u istom programu računamo neke parametre. Nije

korisno svaki puta iznova pisati isti skup BASIC-naredbi jer to zauzima mnogo prostora u memoriji (program je bez potrebe duži) i odnosi mnogo vremena. Da bismo to izbjegli služimo se potprogramima (subrutine).

Potprogram je dio programa napisan tako da ga možemo po volji mnogo puta pozvati iz bilo kojeg dijela programa. U slijedećem primjeru vidimo program u koji unesemo dva broja, a zatim znak da li želimo dijeljenje ili oduzimanje. Program bez obzira kojim redom unesemo brojeve oduzima manji broj od većega, odnosno dijeli veći broj s manjim.

```

10 INPUT "UNESI DVA BROJA ";A,B
20 INPUT " - ILI / ";A$
30 IF A$="-" GOTO B0
40 IF A$(">"/" GOTO 20
50 GOSUB 110
60 PRINT "KVOCIJENT JE ";A/B
70 GOTO 10
80 GOSUB 110
90 PRINT "RAZLIKA JE ";A-B
100 GOTO 10
110 REM POTPROGRAM
120 IF A>B GOTO 160
130 T=A
140 A=B

```

150 R=T
160 RETURN

Program je vrlo nalik na primjer koji smo već imali. Novo je u njemu uvođenje naredbi GOSUB u linijama 50 i 80 i naredbe RETURN u liniji 160.

Naredba GOSUB ima vrlo sličan efekt kao i naredba GOTO. Jedina je razlika što »orač« kada naiđe na naredbu GOSUB »zapamti« broj linije u koju je naišao na ovu naredbu, pa tek onda izvede skok. Tak se zatim nastavlja od linije na kojoj je izveden skok i teče normalno sve dok »orač« ne naiđe na naredbu RETURN. Naredba RETURN znači »vraća se« i naređenje je računalo da se vrati na mjesto odakle je pozvano s naredbom GOSUB. Da bi se vratio »orač« uzme broj linije koji je prethodno zapamti i nastavlja prvom linijom ispod.

Vidimo da smo upotrebom ove naredbe u našem primjeru izbješli potrebu da dva puta ispisujemo linije 120 do 150. Ako program pisemo tako da za svaki od istih poslova koje treba obaviti, napišemo polprogram, a zatim napišemo glavni program koji ne poziva te polprograme, dobit ćemo strukturiran program. Strukturiran programi mnogo su lakši za pisanje i bitno pregledniji: što je najvažnije, svaki od polprograma možemo nezavisno testirati. To je razlog da polprograme pišemo čak i onda kada ih pozivamo samo jednom. Ako polprograme pravilno dokumentiramo, možemo ih čuvati odvojeno i zatim kada radimo novi

program jednostavno u njega uključimo već gotove polprograme. Na primjer polprogram za crtanje e-pse možemo jednako dobro iskoristiti u programu za crtanje kao i u programu za učenje geometrije.

Svaki program u načelu sadržava sva obilježja programa. Zbog toga polprogram možemo raditi jednako kao što smo radili program. Dakle, za njega možemo izraditi poseban opis toka, potrebne dijagrame i tek nakon što ga upišemo i testiramo možemo ga uvrstiti u glavni program.

Posvetimo se malo linijama 130 do 150. Reći smo da bez obzira kojim redom unesemo brojeve moramo oduzeti manji od većega. Zbog toga u liniji 120 testiramo da li je A veći od B. Ako jeste, sve je u redu, ali ako nije moramo sadržaj koji se nalazi u varijabli A staviti u varijabli B, a sadržaj iz varijable B staviti u varijabli A. Jednoličnije moramo zamijeniti varijable. Zamislite da imate dvije boce. U jednoj se nalazi voda, a u drugoj vino. Jasno je da istovremeno ne možemo pretresti vino u onu bocu u kojoj je voda i vodu iz te boce u onu u kojoj je bilo vino. Zbog toga ćemo se poslužiti trećom bocom i privremeno u nju prelićemo vino ili vodu. Slično ćemo postupiti kada moramo zamijeniti vrijednosti dvije varijable. Najprije ćemo u treću varijablu (u našem primjeru T) smjestiti vrijednost jedne od dvije varijable koje mijenjamo. Nakon toga ćemo u tu varijablu smjestiti vrijednost druge varijable, a potom sadržaj druge varijable uzeti iz one privremeno zapamtiše taj postupak jer se vrlo često primjenjuje u

programiranju a pogotovu u sortiranju
Upišite program:

```
10 A=A+1
20 PRINT "POZIV BROJ";A
30 GOSUB 10
```

Ovaj program poziva sam sebe. To znači da »orač«
sistno pamti adresu s koje je pozvan potprogram. To
je vrlo ozbiljna greška, zbog toga su konstruktori »orač«
ograničili broj poziva na 26. Vidimo da je »orač«
kad smo pokušali pozvati potprogram dvadeset i sedmi put
prijavio grešku. Mi možemo iz jednog potprograma po-
zvati drugi, iz ovoga treć, i tako dvadeset i šest puta.
Kazemo da se u potprogramima može ući u dubinu od
dvadeset i šest nivoa. To ograničenje neka vas ne
zabrinjava jer u praksi vrlo rijetko treba više od četiri
nivoa potprograma. Prema tome, dvadeset i šest nivoa
je znatno više nego što će vam ikada zatrebati.

Da ne bi bilo zabune oko tog broja dvadeset i šest,
sve možemo promatrati i ovako. »Oráč«
pamti dvadeset i šest povratnih adresa. Svaki put
kada pozovemo neki potprogram »orač«
zapamti jednu adresu. Svaki put kada se vratimo iz potprogra-
ma, »orač«
zaboravi jednu adresu i oslobodi si pro-
stor za pamćenje nove adrese. Prema tome, u jednom
programu možemo imati koliko god želimo poziva pod
uvjetom da ne idemo u više od dvadeset i šest nivoa.

Kada »orač«
nađe na naredbu RETURN a da prije
toga nije poslan u potprogram tada je sasvim sigurno
u pitanju greška i radi toga će nas on obavijestiti tok-
stom.

7RG GREŠKA

Ispisat će nam u kojoj liniji je našao na RETURN.
Takva je greška vrlo česta kada napišemo potprogram
u nastavku glavnog programa. Tada računar nakon
što je završio izvođenje glavnog programa nastavi rad
u potprogramu i »shvati«
to tek kad nađe na naredbu
RETURN. Zato je najbolje glavni program završiti na-
redbom END pa greške neće biti.

Za naredbu GOSUB vrijedi sve ono što smo rekli za
naredbu GOTO. Prema tome naredba GOSUB može
se nalaziti u naredbi IF, jedino što je tu potrebno
ispred naredbe GOSUB napisati riječ THEN. Pogle-
dajmo to na primjeru:

```
10 INPUT A
20 IF A=1 THEN GOSUB 100
30 IF A=2 THEN GOSUB 200
40 IF A=3 THEN GOSUB 300
50 GOTO 10
100 PRINT "JEDAN"
110 RETURN
```

```

200 PRINT "DVA"
210 RETURN
300 PRINT "TRI"
310 RETURN

```

Vidimo da smo u ovom primjeru pozvali tri različita potprograma u ovisnosti o vrijednosti varijable A. Za svaku vrijednost morali smo upisati po jednu naredbu IF. To nije problem kada radimo s tri, četiri ili čak deset potprograma. Međutim, ako treba pozvati dvadesetak ili više potprograma, morali bismo upisati velik broj IF naredbi. Zbog toga su konstruktori mikračunara «orač» predviđali naredbu koja nam omogućuje da ovo testiranje i skok izvedemo u samo jednoj liniji. Pogledajmo kako bi naš prethodni primjer izgledao ako upotrijebimo tu naredbu.

```

10 INPUT A
20 ON A GOSUB 100,200,300
30 GOTO 10
100 PRINT "JEDAN"
110 RETURN
200 PRINT "DVA"
210 RETURN
300 PRINT "TRI"
310 RETURN

```

U liniji 20 vidimo naredbu ON A GOSUB 100,200,300. Ova naredba uzme vrijednost varijable koja je u njoj navedena i skoči na traženi potprogram. U naredbi 20 navedeni su brojevi 100, 200 i 300. Ako varijabla A ima vrijednost jedan, uzet će se prvi broj iz popisa. Ako je A dva, uzet će se drugi broj itd. Ako A ima vrijednost manju od jedan ili veću od broja brojeva u listi, zvest će se sljedeća instrukcija, za naredbe ON.

Ako varijablu u naredbi ON ima decimalnu vrijednost, računalo će najprije prikazati INT od te varijable. Dakle, za svaki broj veći ili jednak jedan a manji od dva, «orač» će skočiti na prvi broj naveden u listi.

U istoj naredbi može stajati i komanda GOTO umjesto GOSUB. Onda je dozvoljena je formulacija

```
20 ON A GOTO 100,150,200
```

Naredba ON je vrlo moćno «oruđe» za rad jer je moguće uz minimalne izmjene potpuno izmijeniti tok programa. Dovoljno je promijeniti brojeve navedene u naredbi ON.

Najveća vrijednost koju možemo navesti u varijabli koja određuje o skoku u naredbi ON je 255, odnosno u naredbi ON možemo navesti napise 255 adresa. Ako je vrijednost varijable veća od 255 ili manja od nule računalo će prijaviti grešku. Zbog toga je ponekad potrebno prije naredbe ON s naredbom IF testirati ove

vrijednost. Ako je broj izvan predviđenih granica izvesti skok na prvu liniju ispod naredbe ON. Time smo postigli isti učinak kao da naredba ON prihvaća broj koju vrijednost u varijabli.

S obzirom na to da je maksimalna dužina jedne linije u BASICU 72 znaka može nam se desiti da nam ne stanu sve vrijednosti koje želimo napisati u naredbi ON. Tada ćemo upotrijebiti dvije naredbe ON. U prvoj ćemo navesti onoliko adresa koliko nam stane, zatim ćemo u narednoj liniji varijablu koja odlučuje o skoku umanjiti za onoliko koliko smo adresa naveli u prvoj naredbi ON. Nakon toga staviti ćemo novu naredbu ON u kojoj ćemo normalno nastaviti radu. To je ilustrirano u narednom primjeru.

```
10 INPUT A
20 IF A<1 GOTO 10
30 IF A>255 GOTO 10
40 ON A GOTO 100,200,300
50 A=A-3
60 ON A GOTO 400,500,600
70 REM NASTAVAK PROGRAMA
```

Ovaj program nije kompletan i napisan je jedino zato da posluži kao primjer. U naredbi 40 moglo se navesti svih šest adresa, ali sam želio da vidite kako se nastavlja ista adresa u naredbama ON.

PODACI UNUTAR PROGRAMA

U nekim programima potrebno je nakon što se program pokrene, postaviti nekoliko varijabli koje će nam služiti u toku programa. Možemo varijable definirati naredbom za pružanje vrijednosti, ali ako je riječ o većem broju varijabli to će značajno povećati dužinu programa. Ako trebamo na početku programa definirati pet varijabli možemo to učiniti ovako:

```
10 A=1
20 B=5
30 C=7
40 PI=3.141
50 X=255
60 PRINT A,B+C,X*PI
```

Vidimo da smo za definiranje pet varijabli utrošili pet BASIC-linija. Isti primjer možemo napisati i ovako:

```
10 READ A,B,C,X,P1
20 PRINT A,B+C,X*P1
30 DATA 1,5,7,255,3.141
```

Ovdje smo dobili znatno kraći program. To smo postigli upotrebom naredbe READ. Kada računalo naiđe na ovu naredbu, potraži prvu naredbu DATA iz nje pročita brojeve koje pridruži varijablama navedenim u naredbi READ. Pri tome se prvo, varijabli pridružuje prva vrijednost iz naredbe DATA, drugoj druga i tako dalje.

Osim što je ovako napisan program kraći, mnogo je lakše promijeniti vrijednosti koje se pridružuju varijablama. Ako u prvom primjeru želimo promijeniti sve vrijednosti, moramo mijenjati sve linije od 10 do 50. U drugom primjeru dovoljno je promijeniti samo onu liniju u kojoj se nalazi naredba DATA.

DATA-naredba je specifična po tome što je sasvim svejedno gdje se nalazi u programu. Kada računalo naiđe na naredbu READ, pretražuje program počevši od prve linije sve dok ne nađe naredbu DATA. Kada tokom izvršenja programa računalo naiđe na naredbu DATA, ono je preskače potpuno jednako kao naredbu REM. Vidimo da naredbu DATA možemo smjestiti bilo gdje u programu, a da to nema utjecaja na izvršavanje

programa. Zbog toga ćemo naredbe DATA smjestiti ili neposredno iza naredbi READ ili na sam kraj programa. Često je praksa da se naredbe DATA smjestaju na kraj programa.

Upišite u računalo ovaj primjer:

```
10 FOR I=1 TO 5
20 READ A,B
30 PRINT A;"+";B;"=";A+B
40 NEXT I
50 DATA 7,5,8,4,5,3
60 DATA 9,1,2,6
```

Vidimo da u ovom primjeru pet puta izvršavamo naredbu READ i da svak put učtavamo po dvije varijable. Isto tako vidimo da su varijabla A i B pridruživane vrijednosti onim redom kojim su navedene u naredbama DATA. Kada se taj program počeo izvršavati računalo je u prvom naletku na naredbu READ potražilo prvu naredbu DATA i naišlo je u liniji 50. Zato su u prvom prolasku varijablama A i B pridružene vrijednosti 7 i 5. Računalo je nakon što je pročitalo ove dvije vrijednosti, »ZAPAMTILO« koliko je brojeva iz te naredbe DATA pročitalo. U sljedećem naletku na naredbu READ čitanje je nastavljeno iz iste naredbe DATA, počevši od prvog broja koji do sada nije pročitan. Kada se nakon trećeg prolaska iscrpila naredba DATA

u liniji 50 »orao« je automatski potražio prvu slijedeću naredbu DATA i čitanje nastavio iz nje. Kažemo da računalo posjeduje pokazivač koji pokazuje koji je slijedeći podatak u naredbama DATA na redu za čitanje. Svaki put kada se pročita jedna vrijednost pokazivač se premješta na slijedeći podatak. Taj pokazivač nazivamo još i data pointer.

Pri prelasku iz jedne linije DATA u drugu treba biti oprezan. Svi podaci koje čita jedna naredba READ moraju biti u istoj naredbi DATA. Nije dopušteno da naredba READ pročita nekoliko podataka iz jedne naredbe DATA pa da čitanje nastavi u drugoj. Na primjer, ako odjednom čitamo tri podatke, tada broj podataka u naredbi DATA mora biti najmanje tri, odnosno mora biti najmanje još dva podatka iza podatka koji pokazuje pokazivač. Ako u petljici čitamo po tri podatka odjednom, broj podataka u svakoj naredbi DATA mora biti djeljiv sa tri (3, 6, 9, 12...). Ako pri tome pogrešimo, računalo će nam javiti da nema dovoljno podataka.

Upišite sada ovaj program:

```
10 READ A,B
20 PRINT A,B
30 READ A,B
40 PRINT A,B
50 DATA 1,2
```

Kada pokrenemo program, računalo će nailaskom na prvu naredbu READ pročitati dvije vrijednosti iz naredbe DATA – pokazivač će pokazivati kraj te naredbe. Kada se nađe na slijedeću naredbu READ, računalo će prijaviti grešku, jer nema više podataka u naredbi DATA. Dakle, pri pisanju podataka programa moramo paziti da nam se ne desi da čitamo više podataka nego što ih u naredbi DATA ima. Napišite sada prethodni primjer ovako:

```
10 READ A,B
20 PRINT A,B
30 RESTORE
40 READ A,B
50 PRINT A,B
60 DATA 1,2
```

Vidimo da se taj program ispravno izvršava i vidimo da su dva puta čitan isti podaci koji se nalaze u liniji 60. To se desilo zbog naredbe RESTORE. Nakon što je u liniji 10 zvedena naredba READ, pokazivač je pokazivao kraj naredbe DATA. Kada je »orao« naišao na liniju 30 izvršio je naredbu RESTORE koja pokazivač postavlja na početak prve naredbe DATA u programu. Dakle, naredba RESTORE omogućuje nam da podatke koji se nalaze u naredbama DATA čitamo po volji mnogo puta unutar jednog programa.

```

10 READ A$,A
20 PRINT A$;A
30 DATA "ORAO",102

```

Iz ovog primjera vidimo da se string-varijable mogu čitati u naredbama READ jednako kao i numeričke varijable. Vidimo da je dozvoljeno u istoj naredbi DATA navesti i tekstualne i brojčane podatke. Važno je jedino da ne pogriješimo u redoslijedu čitanja, odnosno da ne pokušamo u numeričku varijablu smjestiti tekstualni podatak. Ako to učinimo računalo će prijaviti grešku. Tekstualni podaci koje navodimo u naredbi DATA mogu se pisati ili s navodnicima ili bez njih. Navodnici su nam potrebni jedino kada unutar teksta želimo upotrijebiti zarez, jer u protivnom će računalo zarez shvatiti kao kraj teksta.

Povremeno nam se u radu javi potreba da u sredini programa obrišemo sve varijable. Za to nam služi naredba CLEAR. Kada naiđe na ovu naredbu računalo će «zaboraviti» sve vrijednosti pridruženo varijablama. To znači da će sve varijable nakon ove naredbe imati vrijednost nula, a sve string-varijable dužinu nula.

S upotrebom ove naredbe treba biti oprezan jer su vrlo osjetljivi trenuci kada sa sigurnošću možemo obrisati sve varijable. Na primjer, ova varijabla nije moguće upotrijebiti unutar petlje FOR...NEXT jer će računalo izgubiti vrijednost kontrolne varijable i prijaviti grešku kada naiđe na naredbu NEXT. Petlju koja je naprav-

ljena pomoću naredbe IF naredba CLEAR će najvjerojatnije pretvoriti u beskonačnu petlju.

NUMERIČKE FUNKCIJE

S nekim funkcijama već smo se upoznali (INT, RND). Sada ćemo se pozabaviti ostalim numeričkim funkcijama koje posjeduje «orao». Da ponovimo funkcije nisu naredbe i ne mogu samostajno stajati u BASIC-fiji. Prema njima se odnosimo kao prema bilo kojem matematičkom izrazu. Svakoj funkciji slijedi jedan podatak naveden u zagrad (to je argument funkcije), a funkcija daje određen rezultat (rezultat funkcije). Sve funkcije koje imaju numerički argument i daju numerički rezultat nazivamo numeričkim funkcijama.

Argument funkcije moramo uvijek navesti u zagradu, u protivnom «orao» neće razumjeti izraz koji smo napisali.

Najjednostavnije funkcija u ovoj skupini je funkcija `FRE (X)`. Unutar zagrade kao argument funkcije možemo navesti bilo koju vrijednost. Slično kao i s funkcijom `AND` ova vrijednost ne utječe na rezultat funkcije. Rezultat ove funkcije je slobodan memorijski prostor u bajtovima koji je preostao u «orlu». To znači da ako napišemo

```
PRINT FRE (8)
```

«orao» će ispisati koliko slobodnih bajtova imamo u memoriji.

Ako imate nek program u memoriji obrišite ga i pogledajte koliko ima slobodne memorije. Sada upišite bilo koju BASIC-iniju i ponovno pogledajte koliko ima slobodne memorije. Kao što je i logično, kolikina slobodne memorije smanjuje se sa svakom slijedećom unesenom linijom. Kratim eksperimentiranjem steći ćete sliku o tome koliko memoriju zauzimaju vaš programi. Vidjet ćete da nakon vrlo dugih programa u memoriji ostaje dosta mjesta. Funkcija `FRE` poslužit će prilikom razvoja programa, pogotovu u programima koji opunjavaju s velikim brojem podataka da vidimo kako se troši memorijski prostor. Uz pomoć ove funkcije možemo «zmjeriti» koliko je podataka uneseno u neki program koji radi s podacima. Naprije ćemo napisati cijeli program, zmjeriti koliko je memorije preostalo a zatim ćemo u toku izvršenja programa od tog broja oduzeti rezultat funkcije `FRE` u tom trenutku. Razlika označava memorijski prostor koji su zauzeli podaci.

Druga vrlo jednostavna numerička funkcija je funkcija `SGN`. Ona nam služi da ispitamo kakav predznak ima neki broj. Ako je argument ove funkcije pozitivan broj, ona će dati rezultat 1. Za negativan argument rezultat će biti -1, a za nulu 0. Dakle, `PRINT SGN (5)` dat će rezultat jedan, `PRINT SGN (-5)` rezultat minus jedan a nulu ćemo dobiti jedino ako je argument 0 `PRINT SGN (0)`.

Svoju primenu funkcija `SGN` nalazi u raznim testiranjima. Na primjer nije moguće zračunavati kvadratni korijen iz negativnog broja. U programima u kojima radimo s kvadratnim korijenom možemo upotrijebiti ovu funkciju da testiramo da li je broj negativan.

Funkcija `ABS` ima mnogo širu primenu od funkcija o kojima smo do sada govorili. Ona izračunava apsolutnu vrijednost broja. Najjednostavnije je reći da ova funkcija pretvara predznak broja u plus. To znači: ako je argument funkcije `ABS` negativan broj, dobit ćemo isti taj ali pozitivan broj. Funkcija `ABS` nema nikakav učinak na pozitivne brojeve. Možemo napisati

```
ABS (5) = ABS (-5) = 5
```

Funkcija `ABS` najčešće nam služi kada radimo s kvadratnim korijenom ili logaritmima. Svaki put kada želimo upotrijebiti neku funkciju koja ne dozvoljava negativan argument možemo pomoću funkcije `ABS` učiniti sve argumente prihvatljivima.

vedina od najčešćih operacija u matematičkim formulama je korišćenje. U većini većin potreban nam je kvadratni (drugi) korijen nekog broja. Zbog toga su autori BASICA u BASIC uveli specijalnu funkciju koja vadi drugi korijen iz argumenta. To je funkcija SQR. O toj funkciji nije potrebno mnogo govoriti - dovoljno je reći da argument funkcije ne smije biti negativan. Upotrebom ove funkcije možemo napisati formulu po kojoj izračunavamo hipotenuzu pravokutnog trokuta iz poznatih kateta (Pitagorin poučak) ovako:

$$C = \text{SQR} (A * A + B * B)$$

Ost među vama koji nisu učili trigonometrijske funkcije i logaritme, odnosno oni kojima rječi sin, cos, tan, log ništa ne govore, mogu slobodno preskočiti ostatak ovog poglavlja i odmah prići na sljedeće. Ovdje se nećemo baviti objašnjavanjem trigonometrije i logaritmiranja već ćemo pokušati kako se to radi u BAS-ICU.

«Orao» posjeduje funkciju LOG koja daje prirodni logaritam argumenta. Prirodni logaritam je logaritam broja s bazom e (približno 2.71828). Logaritam je dan s nešto manjom točnošću nego što «orao» navede račun. Međutim, točnost od pet decimalnih znamenki u potpunosti zadovoljava sve potrebe koje se javljaju za pisanje programa. Argument funkcije LOG mora biti pozitivan broj veći od nula.

Iz prirodnog logaritma možemo izračunati broj (antilogaritmirati) pomoću funkcije EXP. To je funkcija koja u biti izračunava e^x na x. Ovu funkciju upotrebjavamo u paru s funkcijom LOG za izračunavanja u kojima su nam potrebni logaritmi.

PRINT EXP (1)

Isprati će nam vrijednost broja e onako kako je «pamti» «orao».

Od trigonometrijskih funkcija «orao» posjeduje funkcije SIN, COS, TAN i ATN koje odgovaraju sinusu, kosinusu, tangensu i arkus tangensu kuta. Vrijednosti koje se navode kao argumenti funkcija SIN, COS, TAN kao i rezultat funkcije ATN su u radijanima.

Ako imamo vrijednosti u stupnjevima možemo ih preračunati u radijane tako da vrijednost u stupnjevima podijelimo sa 180 i pomnožimo sa PI.

kut u stupnjevima * 180 * PI = kut u radijanima

Ako želimo radijane pretvoriti u stupnjeve, podijelićemo vrijednosti sa PI i pomnožili sa 180.

kut u radijanima * PI * 180 = kut u stupnjevima

i znakovi. Svakom znaku pridružen je jedan broj. Na primjer, velikom slovu A odgovara broj 65. Funkcija ASC stampat će nam broj koji odgovara prvom znaku u stringu koji je argument funkcije ASC. Pokušajte

PRINT ASC ("A")

Sada kad smo shvatili kako »orać« pamti znakove, možemo shvatiti kako uspoređuje stringove. Ako u uvjetu unutar naredbe IF napisemo AS=BS, tada »orać« neće uspoređivati dužine stringova, kako bi netko pomislio. On će uzeti prv. znak stringa A i prvi znak stringa B, te usporediti njihove kodove. Ako su različiti, testiranje će se završiti, a ako su isti usporedit će se drugi znakovi u svakom stringu, i tako sve do kraja. Slova su poredana po abecedi (slovo A ima kod 65, slovo B 66 i tako dalje), tako da će string koji po abecedi dolazi iza stringa A bit već od stringa B. Zbog toga što izraz veći i manji mogu dovesti do zabuno u usporedbi stringova umjesto izraza veće uzimamo »raz slijedi« a umjesto »zraza manje« izraz prethodi. Kažemo da string »AAA« prethodi stringu »BB«, odnosno da string »ZZZ« slijedi stringu »EFG«.

U tablici 2 ispisani su svi znakovi koje posjeduje mikroračunalo »orać« i njihovi kodovi. Funkcija ASC dat će broj, koji u tablici odgovara pojedinom slovu.

Argument funkcije ASC dakako, mora biti u zagradama, a ako je riječ o tekstu, on mora unutar zagrada biti u navodnicima. Nije dozvoljeno tražiti ASC od puzavog stringa.

STRING-FUNKCIJE

Za razliku od numeričkih funkcija, string funkcije mogu imati više argumenata. Osim toga, string-funkcije imaju i string kao argument ili daju string kao rezultat.

Prva funkcija string kojom ćemo se pozabavit jest ASC. Argument ove funkcije je string koji ne smije imati dužinu nula. Ova funkcija dat će broj koji je kod prvog karaktera u stringu. Da bismo to shvatili moramo proučiti kako »orać« pamti slova i znakove.

Rekli smo već da je memorija mikroračunala podijeljena u bajtove. U jednom bajtu može se nalaziti broj u rasponu od 0 do 255. Svakom slovu u mikroračunalu odgovara jedan broj. Brojevi od 0 do 31 rezervirani su za specijalni znakove. Od 32 do 127 naznače se slova i znakovi koji postoje u mikroračunalu. Brojevi od 128 do 159 rezervirani su za znakove koje korisnik može sam definirati. Od 160 do 255 naznače se inverzna slova

Suprotan efekt funkciji ASC ima funkcija CHR\$. Ova funkcija dat će za rezultat string dužine jednog znaka koj sadržava znak čiji je kod bio argument funkcije. Na primjer:

PRINT CHR\$ (65)

Ispisat će na ekranu slovo A. Osim pretvaranja kodova u stringove ova funkcija na mikroracunalu - orao- ima neke mnogo značajnije primjene. Rekli smo da znakovi čiji su kodovi od 0 do 31 imaju posebnu primjenu. Ispisivanje nekog od tih znakova može imati utjecaja na izvođenje programa. Ove znakove možemo unijeti direktno sa tastature pritiskom na tipku (CTL) i neko slovo. To možemo učiniti u komandnom kodu. Ako želimo upotrijebiti neki od ovih znakova unutar programa poslužiti ćemo se funkcijom CHR\$. Na primjer, najčešće upotrebljavamo kontrolni znak 12 koji briše ekran. Ako želimo u programu obrisati ekran napisat ćemo

PRINT CHR\$ (12)

Ovdje su ispisani neki od najčešće upotrebljivanih kontrolnih znakova i njihov efekt.

- 4 - pomiče kursor na vrh ekrana bez brisanja ekrana
- 6 - briše ekran od kursora nadalje

- 7 - (beep) ispisivanje ovog znaka proizvodi kratak zvučni signal
- 8 - pomiče kursor za jedno mjesto lijevo
- 9 - pomiče kursor za jedno mjesto desno
- 10 - pomiče kursor za jedan red dolje
- 11 - pomiče kursor za jedan red gore
- 13 - kursor se vraća na početak reda
- 22 - isključuje ili uključuje zvučni signal pri pritisku na tastaturu

Većinu ovih kontrolnih znakova primijenit ćemo kada želimo lijepo urediti ispis na ekranu. Na primjer, ako želimo ispisati neku poruku na početku trećeg reda, ispisat ćemo prvo CHR\$ (12) a zatim dva puta CHR\$ (10). Nakon toga kursor će biti na početku trećeg reda. Znakovi CRS (13) i CHR\$ (10) automatski se ispisuju nakon svake naredbe PRINT koja nije završila zarežom i točka zarežom. Prema tome, ako napismo

PRINT CHR\$ (10)

kursor će se pomaknuti za dva reda. Jednom zato što smo to tražili, drugi put zato što naredba PRINT nije završena zarežom ili točka zarežom. Ako želimo da u pola naredbe PRINT prodemo u nov red stavit ćemo u naredbu PRINT znakove CHR\$ (10) i CHR\$ (13) kao što je to učinjeno u primjeru:

```
10 PRINT CHR$(12); "PRVI RED"; CH
R$(10); CHR$(13); "DRUGI RED"
```

Funkciju CHR\$ iskoristili smo u narednom programu da ispisemo sve znakove kojma mikroračunalo «oro» raspolaže.

```
10 FOR A= 32 TO 255
20 PRINT CHR$(A);
30 NEXT A
```

Vidimo da kontrolnu varijablu A u petlji mijenjamo u rasponu od 32 do 255 jer smo rekli da znakovi od 0 do 31 služe za posebne funkcije.

Naredna funkcija koju često upotrebljavamo pr radu sa stringovima je funkcija LEN. Upišite sljedeći program:

```
10 A$="DUGI STRING"
20 PRINT LEN(A$)
```

Kada pokrenemo ovaj program, vidimo na ekranu ispisani broj 11. To je dužina A\$ koji je argument funkcije LEN. Ova funkcija daje nam broj koji je jednak broju

znakova stringa koji je naveden kao njen argument. Ako u zagradi navodimo tekst umjesto mena stringa, tada taj tekst mora biti u navodnicima. Navodni znaci ne računaju se u dužinu stringa. Funkciju LEN najčešće upotrebljavamo u kombinaciji sa nekom od narednih funkcija.

Funkcija LEFT\$ je prva u nizu funkcija koje imaju više argumenata. U funkciji LEFT\$ navodimo dva argumenta od kojih je prvi string a drugi broj. Oba argumenta navodimo u zagradi a odvajamo ih zareзом. Ova funkcija daje za rezultat string koji je sastavljen od n znakova počevši od lijeve strane stringa koji je naveden kao argument. Pogledajmo to na primjeru

```
10 A$="MIKRORAČUNALO"
20 PRINT LEFT$(A$,5)
```

Vidimo da taj program ispisuje riječ MKRAO, odnosno uzima prvih pet slova iz stringa A. Ako navedemo dužinu koju uzimamo veću od dužine izvornog stringa funkcija LEFT\$ uzet će cijeli izvorni string. Ako navedemo dužinu manju od 1 računalo će prijaviti grešku. Kao string-argument funkcije možemo navesti i tekst koji tada mora biti u navodnicima.

Funkcija RIGHT\$ ima iste argumente kao i funkcija LEFT\$ i također formira podstring izvornog stringa dužine n znakova, jedino što se znakovi ovaj put uzimaju s desne strane izvornog stringa. U sljedećem primjeru

uzeli smo osam znakova s desne strane izvornog stringa

```
10 A$="MIKRORAČUNALO"
20 PRINT RIGHT$(A$,8)
```

Ako je navedena dužina manja od 1 «brak» da pojavi grešku, a ako je veća od dužine izvornog stringa, uzat će cijeli izvorni string. U svim funkcijama koje operiraju sa stringovima numerički argumenti ne smiju biti veći od 255 jer je i maksimalna dužina stringa ograničena na 255.

Funkcija MID\$ ima tri argumenta. Prvi argument je string (izvorni string), a druga dva su numerički argumenti. Ova funkcija daje podstring izvornog stringa počevši od znaka m dužine n. U narednom primjeru iz izvornog stringa zvadili smo podstring koji počinje na šestom znaku i dugačak je pet znakova.

```
10 A$="MIKRORAČUNALO"
20 PRINT MID$(A$,6,5)
```

Prvi numerički argument ne smije biti manji od jedan, a ako je veći od dužine izvornog stringa, kao rezultat ćemo dobiti prazan string. Drugi numerički argument ne smije biti manji od nule. Ako je on nula, dobit ćemo

prazan string. Ako je drugi numerički argument veći od preostale dužine izvornog stringa, dobit ćemo desni dio izvornog stringa počevši od znaka koji je odredio prvi numerički argument.

Funkciju MID\$ možemo upotrijebiti da pretražimo neki string i ustanovimo da li se u njemu nalazi određen znak ili nek drugi string. Naredni program analizira string A i traži u njemu prisutnost stringa B. Ako je drugi numerički argument u funkciji MID\$ jedan, istim programom možemo tražiti prisutnost određenog znaka u stringu.

```
10 A$="MIKRORAČUNALO"
20 B$="ORA"
30 FOR A=1 TO LEN(A$)-LEN(B$)
40 IF B$=MID$(A$,A,LEN(B$)) THEN
50 PRINT A
```

Program nam ispisuje adresu (ili više adresa ako se string pojavljuje više puta) podstringa znaka podstringa B u stringu A.

Ponekad je potrebno da broj koji imamo u nekoj numeričkoj varijabli pretvorimo u string. To radimo zato da možemo na neki string pripisati (navodezat) broj ili zato da utvrdimo na spis na primjer zbog potpisivanja broja. Pri tome upotrebljavamo funkciju STR\$ koja ima

jedan numerički argument. Funkcija STR\$ daje za rezultat string koji izgleda onako kako bi izgledao spis broja na ekranu.

Ako zelimo pravilno potpisivati brojeve, možemo broj pretvoriti u string, zatim izmijeniti «dužinu» broja i pravilno potpisati broj. Kako se to radi prikazano je u narednjem programu koji ispisuje brojeve od 1 do 120 sa korakom 9 i pravilno ih potpisuje.

```
10 P$=""
20 FOR A=1 TO 120 STEP 9
30 A$=STR$(A)
40 PRINT LEFT$(P$,5-LEN(A$));A$
50 NEXT A
```

Potpisivanje smo izveli tako da smo izmijenili dužinu stringa A\$ koji je jednak broju A, a zatim spisati onoliko praznih mesta «spred» broja koliko je manjkalo da broj izgleda sa vidljivom dužinom dohite dužine 5. To smo postigli tako da smo ispisali LEFT\$ stringa P dugacak onoliko koliko nam je mjesta manjkalo. Prethodno smo stringu P delimitirali kao string koji sadržava beskrajno 0-ve. Ova metoda možemo primeniti za pravilno potpisivanje decimalnih brojeva, jedino što tada moramo izmisliti dužinu samog eksponentnog dijela. Takođe možemo tako da prethodno izmisliti dužinu zvanog broja. Pri potpisivanju negativnih brojeva treba u vidu predznak minusa koji će povećati dužinu broja.

Suprotan učinak od funkcije STR\$ ima funkcija VAL. Ona pretvara tekst stringa u numeričku vrijednost. Dakako, sadržaj stringa mora biti numerički. Dovoljna je upolena znamenka od 0 do 9 i do nimala ne loške. Ako funkcija VAL nađe na nenumerički podatak (znak ili slovo) dat će rezultat nula. Ako nađe na numerički podatak pa za njega nenumerički uzat će kompletan dio stringa do prvog nenumeričkog podatka i pretvoriti ga u broj. Rezultat će biti nula i ako string ima dužinu nula.

PRINT-FUNKCIJE

U više navrata već smo govorili o naredbi PRINT. Ako pogledate primjere koje smo do sada obradili u ovoj knjizi, vidjeće da svakom program sadržava ba-

rem jednu naredbu PRINT. Da bi mogućnosti u radu s ovom naredbom bile još veće, konstruktor -orič- ugradio su tri funkcije koje rabimo u vezi s naredbom PRINT.

Funkcija POS je zapravo klasična numerička funkcija. Ima jedan numerički argument koji, kao i u funkciji FRE, ne utječe na rezultat funkcije. Dakle, u zagradi možemo navesti bilo koji broj. Rezultat ove funkcije je broj koji nam govori na kojem se mjestu (na kojem slovu) u redu nalazi kursor. Prvi znak u redu (početak reda) ima vrijednost nula. Najveći znak je 71 jer smo rekli da je maksimalna dužina jedne linije 72 znaka. Ako pri inicijalizaciji računala na pitanje DULJINA LINIJE? odgovorimo s nekim drugim brojem, na primjer 50, tada će se promijeniti i gornja granica broja koji će nam davati funkcija POS (49). Ovdje govorimo o tzv. logičkoj liniji, za razliku od fizičke linije koja uvijek ima 32 znaka jer «orao» na ekranu ispisuje 32 znaka u jednom redu. Napišite program:

```
10 PRINT POS(0);
20 GOTO 10
```

Vidimo da nam «orao» ispisuje rastuću niz brojeva sve do vrijednosti 71, a zatim počinje iznova. To su brojevi koji prikazuju adresu kursora unutar logičke linije. Ako u našem primjeru zadržimo na kraju linije 10 točka

zarez, «orao» će stalno ispisivati vrijednost nula. Zaslou?

Funkcija SPC služi isključivo unutar naredbe PRINT. Funkcija SPC ima numerički argument i ispisuje onoliko praznih mjesta na ekranu koliko je vrijednost argumenta. Prema tome, ako napisemo

```
PRINT SPC(20); "A"
```

na ekranu će nam se ispisati slovo A s dvadeset praznih mjesta ispred njega. Ako kao argument navedemo broj koji je veći od 32, spisivanje će početi u novi red i ostaviti jedan red prazan. Za svakih 32 u argumentu dobit ćemo po jedan prazan red. Na primjer, SPC(74) dat će dva prazna reda i deset praznih znakova u sljedećem redu. Argument koji navodimo u funkciji SPC ne smije biti veći od 255 niti manji od nule. Ako kao argument navedemo nulu, «orao» će to shvatiti kao da smo napisali 255, ispisati 256 slobodnih znakova.

Funkciju SPC možemo upotrijebiti i za razdvajanje dva ispisa na ekranu. Tada ćemo ispisati ovu funkciju između dva teksta koje želimo odvojiti.

Druga funkcija s naredbom PRINT je funkcija TAB. U ovoj funkciji navodimo broj koji od strane na kojem će se mjestu počevši od početka reda ispisati sljedeći ispis. Na primjer

10 PRINT TAB(10); "A"

Štampa se slovo A na desetom mjestu od početka reda. Prvi znak u redu je prazan, to se odražava u tome da se u naredbi mora navesti kao argument ove funkcije najmanji dozvoljeni broj je broj 255.

Funkcija TAB uglavnom upotrebljavamo za tabuliranje, štampa se data i u praznim. Naredba

```
10 PRINT TAB(10); "A"; TAB(20); "B"
```

Štampat će slovo A na desetom mjestu a slovo B na dva deset. Ako je argument funkcije TAB u ovom naredbi od prethodne ova funkcija neće imati nikakav učinak. Ako je argument naveden u funkciji veći od 255 ispis će se nastaviti u novom redu isto kao i u naredbi FOR. U ovom primjeru primjenjujemo funkciju TAB da bi se ispis izveo u načinu sličnom onom na ekranu.

```
10 FOR A=0 TO 6.28 STEP .25
20 B=16+15*SIN(A)
30 PRINT TAB(8); "*"
40 NEXT A
```

Pozite na liniji 20. U njoj izračunavamo vrijednost sinusa. Da bismo ga lakše pokazali na ekranu upotrijebimo funkciju TAB. Imamo prijedlog, raspon vrijednosti ispisati nam daje vrijednosti od -1 do +1. Zbog toga možemo vrijednosti sinusa sa 16. Imamo dobivamo vrijednosti raspona od -1 do +15. Kako u funkciji TAB nisu prihvaćene negativne vrijednosti, moramo na kraju dodati 16 pa dobijemo vrijednosti u rasponu od 1 do 17. Kao funkcije TAB ne može imati prihvata.

U naredbi naredbe PRINT, osim spomenute funkcije, postoji još jedna funkcija koja omogućava nam ispisati na ekranu sve znakove. U naredbi, da ispisujemo znak PRINT možemo u svakom trenutku otkupiti znak znak (?). Sljedeća dva primjera imaju potpuno isti učinak.

```
10 PRINT A
11
10 ? A
```

Ako pri unosu programa unesete naredbu PRINT napravit će uprsk, morat će to praviti naredbu. Ako u ispisati program vidjet ćemo da je znak završetka naredbe PRINT. Znak ispisnika možemo slobodno upotrebljavati pri unosu programa, ali kad pisete program na papiru, pisate naredbu PRINT. U protivnom znak stoni neće biti naročito čitljivi.

U naredbi naredbe PRINT, osim spomenute funkcije, postoji još jedna funkcija koja omogućava nam ispisati na ekranu sve znakove. U naredbi, da ispisujemo znak znak PRINT možemo u svakom trenutku otkupiti znak znak (?). Sljedeća dva primjera imaju potpuno isti učinak.

redbe unutar jedne linije odvajamo dvotočkom (,). Dovoljen je ovaj izraz

```
10 A=S : B=7 : PRINT A+B
```

Kao što se pri čitanju normalnog izvoda programa naredbe izvršavaju u redoslijedu prema dozi, tako će se jedna za drugom izvršavati i se naredbi u jednoj liniji slijeva nadasno.

Takvo ispisivanje naredbi krije u sebi neke zamke. Kao prvo, ovako napisan program je vrlo teško čitati. Naredbom GOTO i GOSUB možemo pozvati samo onu naredbu koja nam treba. Prema tome, ako u liniji 10 imamo tri naredbe, ne možemo nikako izvesti skok na drugu naredbu u toj liniji. Naredba GOTO 10 će izvesti skok na prvu naredbu u liniji 10 i tek nakon što izvrši prići na drugu naredbu u istoj liniji.

Za naredbu GOTO nema svrhe navoditi neke druge naredbe jer se one nikada neće izvršiti. Programer kad naiđe na naredbu GOTO skok će učiniti u koju ga ova naredba hoće, tj. nema načina da se vratimo izvršimo naredbe za GOTO i tako nema svrhe navoditi naredbe za GOTO i tako nema svrhe navoditi naredbe. Za naredbu REM. Kad naiđemo na ovu naredbu program ne čita ostatak linije, prema tome ne može biti ni naredbe za REM. Na primjer: REM

Iza naredbi THEN u naredbi IF možemo također navesti nekoliko naredbi odvajajući dvotočkom. Treba imati

na umu da će se sve ove naredbe izvršiti jedino ako je uvjet postavljen u naredbi IF zadovoljen. Ako uvjet u naredbi IF nije zadovoljen, «orać» će jednostavno preskočiti ostatak linije. U sljedećem primjeru vidimo dosta čestih grešaka. Počija FOR - NEXT zatvorena je i za i jed. THEN u naredbi IF zbog toga se ne izvršava pravilno.

```
10 FOR A=1 TO 10
20 PRINT A
30 IF A<5 THEN PRINT "MANJE OD
5" : NEXT A
```

Vidimo da program radi dok je zadovoljen uvjet u naredbi IF i kada uvjet nije zadovoljen, rad programa se prekida. Razlogu ova program treba najmanje dva:

```
10 FOR A=1 TO 10
20 PRINT A
30 IF A<5 THEN PRINT "MANJE OD
5"
40 NEXT A
```

Ovoj funkciji dal' smo ime K ili je ono slovo iz riječi FN) - relativni argument A. Ako želimo pozvati ovu funkciju učinićemo to ovako:

20 PRINT FNK (5)

Prethodna linija ispisat će nam kvadrat broja 5. U pozivu funkcije možemo kao argument navesti bilo broj bilo varijablu ili matematički izraz.

Flexi smo da funkciji pridjeujemo relativni argument. Unutar zagrada navodimo ime jedne varijable. To isto ime upotrijebit ćemo s desne strane znaka jednakosti dakle prikom definiranja funkcije. Kada pozovemo funkciju navest ćemo u zagradi neki drugi podatak (broj ili izraz). Vrijednost tog podatka pridjeći će se relativno varijabli navedenoj u funkciji i tada će se izvesti ono što je navedeno u definiciji funkcije. Ova varijabla nema nikakav učinak na ostale varijable - izračunava se od njih nakon što je zaviseno izračunavanje funkcija. Možemo slično u programu imati stvarami varijablu koja ima isto ime kao i relativna varijabla i upotrijebiti relativne varijable neće izmijeniti vrijednost stvarne varijable. Relativne varijable nazivamo još i privremenim ili lokalnim.

Pri definiranju funkcije izraz s desne strane znaka jednakošću možemo uzeti i stvarne varijable iz programa. Jedno je oglašavanje da ne možemo upotrijebiti

KORISNIČKE FUNKCIJE

U prethodna tri poglavlja proučili smo sve funkcije kojima raspolaže -orac-. Osim navedenih postoji i mogućnost definiranja novih funkcija. Takve funkcije nazivamo korisnički definirane funkcije.

Sami možemo definirati isključivo numeričke funkcije koje imaju jedan argument. Rezultat ovih funkcija je dakako, broj. Svakoj funkciji moramo podijeliti ime da bi je kasnije mogli pozvati. Ime funkcije je jedno slovo iz engleske abecede. Nisu dopuštena naša slova Č, Ć, Đ, Š i Ž. Funkciju definiramo naredbom DEF FN.

Za primjer uzećemo definiciju funkcije koja izračunava kvadrat broja

```
10 DEF FNK (A)=A*A
```


stvarnu varijablu koju se zove isto kao i relativna, jer će «orač» shvatiti da želi mo upotrijebiti relativnu.

Funkcija mora biti definirana prije nego što je pozovemo. To znači ako neku funkciju pozivamo u liniji 100, definicija te funkcije mora imati inisk broj manji od 100. Definiranje funkcije ne smije se prilikom završenja programa preskočiti naredbom GOTO. «Orrač» može skonstit. funkciju tek nakon što je prešao liniju u kojoj je ona definirana. U praksi naredbe DEF FN najčešće smještamo na sam početak programa.

Unutar jednog programa možemo istu funkciju dva puta definirati. Prva definicija funkcije primjenjivat će se sve dok računalu ne nađe na liniju u kojoj se predefinirava funkcija. Nakon te linije upotrijebit će se druga definicija.

Funkcije ćemo primjeniti kada često unutar jednog programa moramo izračunavati isti izraz.

POLJE PODATAKA

U nekim programima sresti ćemo se s većom skupinom podataka koji pripadaju nekoj cjelini. Na primjer, moramo izračunati srednju vrijednost temperature u nekom tjednu. Da bismo to izračunali, potrebno nam je sedam podataka koji označavaju temperature zabilježenu u svakom pojedinom danu. To možemo napraviti ovako:

```
10 READ PO,UT,SR,CT,PE,SU,NE
20 U=PO+UT+SR+CT+PE+SU+NE
30 S=U/7
40 PRINT "SREDNJA TEMPERATURA J
E";S
50 DATA 15,17,21,19,18,20,22
```

Ovaj je pristup moguć kada radimo sa sedam podataka. Zamisli da računalu sročnu vrijednost temperature u godini dana. Trebalo bi nam 365 varijabli. Ovaj problem možemo riješiti s potrebom polja podataka. Polje podataka je skupina podataka kojima je pridruženo zajedničko ime. Svaki takav podatak naziva se element polja i odgovara mu jedan broj, odnosno adresa u polju. Možemo reći da su elementi nekog polja indeksne varijable.

Da bismo se poslužili poljem moramo naredbom DIM definirati u memoriji prostor za polje. Ime polja formira

se isto kao ime varijable. Pogledajmo naš primjer riješen upotrebom polja:

```
10 DIM A(6)
20 FOR I=0 TO 6
30 READ A(I)
40 S=S+A(I)
50 NEXT I
60 PRINT "SREDNJA VRIJEDNOST JE
";S/7
70 DATA 15,17,21,19,18,20,22
```

Vidimo da smo dobili program koji je nešto duži od programa u prvom primjeru. Ali ova, program ostaje isto toliko velik bez obzira koliko velik broj podataka obrađivati. Dovoljno je samo promijeniti vrijednosti u naredbama koje određuju dimenziju polja i broj izvršenja petlje.

Vratimo se naredbi DIM. Naredba DIM u liniji 10 definira polje A koje ima sedam elemenata. To su elementi A(0), A(1), A(2), ..., A(5) i A(6). Svaki od ovih elemenata možemo upotrijebiti isto kao i obične varijable. Broj u zagradi indeks možemo zamijeniti varijablom što vidimo u prethodnom primjeru i tada možemo elementima polja manje računati u petljanjima. Indeks elemenata polja počinje od nule. Dakle, prvi element polja

ima indeks nula. Posljednji element polja ima indeks jednak broju koji smo naveli u naredbi DIM. Vidimo da će naredba DIM kreirati polje koje ima jedan element više od broja koji je naveden iza naredbe DIM. Ako nas to zbunjuje, možemo raditi s poljima tako da ne rabimo indeks nula, pa će nam polje imati onoliko elemenata koliko smo naveli u naredbi DIM.

Ako u programu upotrijebimo indeks koji je manji od nule ili veći od najveće dimenzije polja, računalo će prijaviti grešku. Maksimalna veličina polja, odnosno maksimalni broj elemenata polja određen je jedino našim pozivom slobodnom memorijom.

Sve što smo do sada rekli za numerička polja vrijedi i za string-polja. Možemo kreirati string-polja elementi kojih su stringovi s indeksom. Proširimo naš prethodni zadatak da nam ispisuje temperaturu u svakom pojedinom danu. Pri tome ćemo uzeti string-polje da ispišemo dane u tjednu.

```
10 DIM A(6) : DIM AS(6)
20 FOR I=0 TO 6
30 READ A(I)
40 NEXT I
50 FOR I=0 TO 6
60 READ AS(I)
70 NEXT I
80 FOR I=0 TO 6
```

```

90 S=S+A(I)
100 PRINT A$(I); TAB(15);A(I)
110 NEXT I
120 PRINT "SREDNJA TEMPERATURA
JE";S/7
130 DATA 15,17,21,19,18,20,22
140 DATA PONEDELJAK,UTORAK,SR
JEDA,CIVRTAK,PETAK,SUBOTA,NEDJ
ELJA

```

Sva polja s kojima smo do sada radili bila su jednodimenzionalna. To znači da je svaki element polja imao samo jedan indeks. Jednodimenzionalna polja možemo zamisliti kao policu na kojoj stoje knjige u jednom redu. Prvoj knjizi odgovarao bi indeks nula, drugoj jedan itd.

Ako uzmemo policu na kojoj knjige stoje u višim nivoa, tada za svaku knjigu moramo reći na kojem nivou stoji i koja je knjiga po redu u tom nivou. Vidimo da bi svakoj knjizi odgovarala dva indeksa. Takvo polje nazivamo dvodimenzionalnim poljem. Kreirali ćemo ga tako da u naredbi DIM za imena polja u zagradama navedemo dva broja odvojena zarezom.

Kako u zagradu navedemo tri broja, dobit ćemo trodimenzionalno polje. U trodimenzionalnom polju svakom elementu polja pripadaju tri indeksa. Na primjer, slova unutar jedne knjige možemo gledati kao ele-

mente trodimenzionalnog polja. Svakom slovu odgovaraju tri broja. Prvi broj govori na kojoj se stranici knjige nalazi slovo, drugi broj govori nam red na stranici a treći broj slova u redu.

Mogu se dimenzionirati i višedimenzionalna polja od četiri, pet ili čak šest dimenzija, ali je pri tome tako kreirano polje u pravilu toliko veliko da mikroročunalo ne raspolaže s dovoljno memorije. Na primjer, polje koje ima šest dimenzija, kojemu je svaka dimenzija deset, zauzimalo bi oko 5 000 000 bajtova odnosno oko 5000 K. Vidimo dakle da je jedino ograničenje prilikom dimenzioniranja polja raspoloživ memorijski prostor.

Dvodimenzionalna i trodimenzionalna polja upotrebljavat ćemo kada imamo dvije skupine istovrsnih podataka pri čemu svakom podatku u jednoj skupini odgovara podatak u drugoj skupini. Tada ćemo upotrebljiti dvodimenzionalno polje kojemu će jedna od dimenzija biti jedan. Uzmimo za primjer da u nekom skladistu treba svakom proizvodu pridružiti jedan broj (skladišni broj) i cijenu. Tada ćemo za 100 proizvoda upotrebljiti polje dimenzionirano ovako:

```
DIM P(99,1)
```

Tako svakom proizvodu odgovara u dva elementa polja. Broj prvog proizvoda smjestiti ćemo u P(0,0) a cijenu stog proizvoda u P(0,1).

Dvodimenzionalna polja poslužile nam pri radu s matricama. Matrice ili tablice su naziv podataka sačinjen od jednakih skupina podataka. Ekran na kojem radimo ispise tekst možemo zamisliti kao dvodimenzionalnu matricu u kojoj svakom slovu odgovara dva indeksa. Prvi određuje broj reda na ekranu, a drugi broj znaka u redu.

Trodimenzionalno polje po nažbe u prikazivanju trodimenzionalnih modea. Na primjer, ako želimo označiti neke točke u prostoru, svakoj točki odgovaraju tri dimenzije: širina, dužina i visina.

Polja s više od tri dimenzije vrlo se nteško primjenjuju. Osim toga, čovjek prilično teško radi s tim poljima jer pri radu ne može zamisliti nekakav prihvatljiv oblik organizacije podataka. To je zbog toga što smo u svakodnevnom životu navikli raditi s modeima koji imaju najviše tri dimenzije.

LOGIČKE OPERACIJE

Svaki broj i znak kojima mikračunalo barata smješteni su u bitove. Jedan bit može biti vrijednost od 0 do 255. Pogledajmo, zašto baš ova vrijednost.

Mikračunalo sve podatke kojima barata i sve operacije koje izvršava prikazuje u obliku električnih impulsa. Električni impulsi mogu biti u dva različita stanja (ima struje ili nema struje). Odnosno, postoji protok elektricne struje kroz neki sklop (sklop vodi) ili ne postoji (sklop ne vodi). Stanje u kojem određeni sklop ne vodi označavamo nulom, a stanje u kojem taj sklop vodi jedinicom. Ova stanja nazivamo logička stanja sklopa, pa kažemo logička nula i stanje logičke nule i logička jedinica. Takav sklop možemo zabilježiti samo opisana dva stanja i takav sklop nazivamo BIT. Dakle, jedan bit ima stanje 1 ili stanje 0.

Ako upotrijebimo dva bita možemo prikazati četiri različita stanja. To su

- oba bita su 0
- prvi bit je 0, drugi 1
- prvi bit je 1, drugi 0
- oba bita su 1

Vidimo da sa dva bita možemo prikazati različite vrijednosti. Takvo prikazivanje brojeva naziva se binarni prikaz brojeva ili prikaz brojeva s bazom dva.

DEC	BIN	HEX
0	= 0000	= 0
1	= 0001	= 1
2	= 0010	= 2
3	= 0011	= 3
4	= 0100	= 4
5	= 0101	= 5
6	= 0110	= 6
7	= 0111	= 7
8	= 1000	= 8
9	= 1001	= 9
10	= 1010	= A
11	= 1011	= B
12	= 1100	= C
13	= 1101	= D
14	= 1110	= E
15	= 1111	= F

U tablici vidimo binarni prikaz svih brojeva od 0 do 15. Vidimo da smo prikazujući vrijednosti od 0 do 15 iskoristili sve moguće kombinacije binarnih brojeva sa četiri znamenke. Možemo reći da sa četiri binarne znamenke možemo prikazati 16 različitih stanja. Možeće je matematički dokazati da četiri binarne znamenke daju dva na četvrtu različitih stanja, odnosno da n binarnih znamenki daje dva na n različitih stanja. Svakoј vrijednosti u prethodnoj tablici pridružili smo jedan broj ili slovo, što vidimo u trećem stupcu tablice. Vrijednostima od

0 do 9 pridružili smo brojeve od 0 do 9, a vrijednostima od 10 do 15 slova od A do F. Tako, svakoј vrijednosti od 0 do 15 pridružili smo jedan znak. Takav način prikazivanja brojeva nazivamo heksadecimalnim sistemom prikazivanja brojeva. On ima bazu 16. Pomoću jedne heksadecimalne znamenke možemo prikazati četiri binarne znamenke.

Jedan bajt, odnosno jedna osnovna jedinica memorije, sastoji se od osam binarnih znamenki. Dakle, svak bajt možemo prikazati pomoću osam jedinica ili nula. Ako «orao» u nekom bajtu ima vrijednost 100, on će u tom bajtu imati sljedeći broj

10000010

Sa osam binarnih znamenki možemo prikazati dva na osmi različitih stanja, odnosno 256 različitih vrijednosti. Kako je jedna od vrijednosti nula, najveći broj koji se može nalaziti u jednom bajtu je 255.

Binarno prikazivanje brojeva je za ljude jako neopredjeljeno i teško čitljivo. Zbog toga stanje pojednog bajta prikazujemo pomoću dva heksadecimalna znaka. Pretvorba iz binarnog u heksadecimalno je vrlo jednostavna. Dovoljno je da uzmemo lijeva četiri bita iz binarnog broja i umjesto njih napišemo odgovarajuću heksadecimalnu vrijednost iz tablice. Sada to isto učinimo s desna četiri bita i dobili smo dvoznamenkasti heksadecimalni broj koji prikazuje stanje određenog bajta. Na primjer

11111111 = FF
 10100101 = A5
 10011110 = 1E

Ako se u heksadecimalnim broju pojavljuje neka vrijednost označena slovom, onda nam je odmah jasno da je u pitanju heksadecimalni broj. Za broj 56 ne znamo da li je riječ o decimalnom broju 56 ili heksadecimalnom broju. Zbog toga ispred heksadecimalnih brojeva obično pišemo malo slovo h ili znak &. Broj 56 heksadecimalno napisat ćemo h56 ili &56.

Na kraju ove knjige imate program koji pretvara brojeve iz binarnih u decimalne ili heksadecimalne, iz decimalnih u binarne ili heksadecimalne i iz heksadecimalnih u decimalne ili binarne.

Bez obzira na to kako je prikazan pojedini broj, sve matematičke operacije dajuće će točan rezultat. S binarnim brojevima možemo izvoditi i određene logičke operacije. To su logičke operacije NOT (ne), OR (ili) i AND (i).

Logička operacija NOT izvest će nverziju, ili negaciju, bita. Prema tome NOT 1 jednako je nula, a NOT 0 jednako je jedan. To obično prikazujemo ovako:

A	NOT A
0	1
1	0

Ako zvedemo operaciju NOT nad nekim brojem, svi bitovi tog broja promijenit će svoju vrijednost. Sve nule postat će jedan, a sve jedinice nula. Na primjer:

NOT 11001010 = 00110101

Logička operacija OR izvodi se između dva broja. Ako izvedemo OR između dva bita rezultat će biti jedan ako je bilo koji od bitova jedan. Prema tome, logička operacija OR daje vrijednost nula jedino ako su oba bita nula. U tablici to izgleda ovako:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Kada izvedemo logičku operaciju OR između dva osmobilna broja dobit ćemo osamobilni broj u kojem su jedinice svi oni bitovi od kojih je barem jedan bit u izvornim podacima 1. Dakle,

11001010 OR 01010101 = 11011111
11110000 OR 10101010 = 11111010

Ako zvedemo OR između nekog broja i nule rezultat će biti isti kao prvi broj, a ako zvedemo OR između nekog broja i hFF (255) rezultat će biti hFF.

Logička operacija AND također se izvodi između dva broja i daje rezultat 1 jedino ako su oba broja 1. To u tablici izgleda ovako:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Logičko AND između dva osambitna broja dat će osambitni broj u kojemu su jedinice oni bitovi koji su u oba izvorna broja jedan.

11001010 AND 01010101 = 01000000
11110000 AND 10101010 = 10100000

Logičko AND između broja A i nule dat će rezultat nula, a logičko AND između broja A i hFF dat će rezultat A.

Osim ovih osnovnih logičkih operacija upotrebljava se još i složena logička operacija XOR ili EOR. Ovu

operaciju nazivamo ekskluzivno ili, a možemo je dobiti prema ovoj formuli:

A XOR B =
A AND NOT B OR NOT A AND B

Ova operacija primenjena na dva binarna broja daje jedan jedino ako je samo jedan od ta dva broja jedan. U tablici to izgleda ovako:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Logičko XOR između dva osambitna broja dat će osambitni broj u kojem su jedinice oni bitovi u kojima je samo jedan od izvornih bitova 1.

11001010 XOR 01010101 = 10011111
11110000 XOR 10101010 = 01011010

XOR broja A i nule dat će broj A, a XOR broja A i hFF dat će NOT A.

LOGIČKE OPERACIJE U BASICU

»Original« BASIC posjeduje logičke operacije NOT, OR i AND. Ove operacije možemo zvoditi na trojevičima, ali na tome moramo biti svjesni da su u pitanju binarne logičke operacije, dakle, zvođe se na binarnim prikazima brojeva u kojima radimo.

Na primjer

```
PRINT 10 OR 3
```

ispisat će broj 11 jer to u binarnom obliku izgleda ovako:

```
00001010 OR 00000011 = 00001011
```

```
PRINT 10 AND 3
```

ispisat će broj 2 jer to u binarnom obliku izgleda ovako:

```
00001010 AND 00000011 = 00000010
```

Vjerojatno će se javiti u programu potreba da logičke operacije zvodimo na brojevima. Mnogo česta primjena ovih operacija je povezivanje uvjeta. U niti na naredbi IF možemo navestioliko ko uvjeta povezan h logičkim operatorima. Ako dva uvjeta povežemo n, e logičkom naredbom AND iza i IF-EN izvest će se jedna, ako su istovremeno oba postavljena uvjeta. Na primjer

```
IF A=5 AND B>3 THEN PRINT "UVJET  
ISPUNJEN"
```

ispisat će UVJET ISPUNJEN jedino ako je A jednako 5 i B veće od 3. Ako bilo koji od ova dva uvjeta nije ispunjen, neće biti ispisani tekst.

Primjena složenih uvjeta znatno nam olakšava programiranje i smanjuje potreban broj naredbi IF u programu. Medutim, složeni logički uvjeti su zbog svoje zamršenosti vrlo česti uzrok grešaka u programu. Zbog toga početnici ne bi trebali pretjerivati s sklapanjem složenih logičkih uvjeta.

GRAFIKA

Jedna od najčešćih upotreba računara u svakodnevnom životu je izrada grafika. Čovjeku je mnogo lakše da sagleda odnose između nekih veličina ako su one prikazane grafički. Time se informacija po čovjeka brie i lakše prepoznaje što je osobito važno za poslove, koji zahtijevaju donošenje mnogo odluka u vrlo kratkom roku.

Ovo, raspoloživo mogućnostima izrada grafika. Za potrebe rada na grafici ekran mora biti svan i kao prvi kvadrant koordinatnog sustava. Donja lijeva točka

ekrana ima koordinate 0,0 a gornja desna koordinate 255,255. Vidimo da je ekran kvadratna ploča s 256 x 256 točkica. Bilo koju od ovih točaka možemo osvjetliti. Za rad s grafikom služit će nam tri BASIC-instrukcije.

Instrukcija PLOT izabire sjede dva broja osvjetliti će na ekranu točku koju adresiraju ta dva broja. Napišite

```
10 PRINT CHR$(12)
20 PLOT 127,127
```

Vidimo da se pojavila jedna točkica na sredini ekrana. Prvi broj označava koordinatu x odnosno za koliko u desno pomakemo točku koju crtamo. Drugi broj je koordinata y odnosno na koju ismrtamo točku. Prema tome 0,255 adresira gornju lijevu, a 255,0 donju desnu točku ekrana.

Instrukcija DRAW pomiče liniju od grafičkog kursora do adrese koju određuju brojevi navedeni za naredbe DRAW.

Proučimo prvo složen grafički kursor. Kao što postoji kursor za tekst koji određuje kam o će se naredba napisan znak tako postoji grafički kursor. Kada uključimo računalu grafički kursor mijenja se u konjarn likom kula ekrana odnosno na adres 0,0. Svaka grafička naredba koja izvršimo pomažeut će grafički kursor. Instrukcija PLOT postavlja prve kursor na adre-

sto na ekranu na kojem nacrtati točku. Instrukcija DRAW smjesti grafički kursor u točku u kojoj je završeno crtanje.

Brojevi koje navodimo iza naredbe DRAW, služe kao i u naredbi PLOT, samo služe da određuju gdje će završiti linija. Prema tome, ako napišemo

```
10 PLOT 0,0
20 DRAW 255,255
```

crtat ćemo diagonalu ekrana. Linija 10 postavlja grafički kursor u donji lijevi kut ekrana, a linija 20 povlači liniju od grafičkog kursora do donjeg desnog kuta ekrana. Upišite program.

```
10 FOR A=0 TO 255 STEP 8
20 PLOT A,0
30 DRAW A,255
40 PLOT 0,A
50 DRAW 255,A
60 NEXT A
```

Vidimo da ovaj program iscrtava kvadratnu mrežu. Promjenom vrijednosti za riječ STEP u liniji 10 možemo dobiti mrežu različite gustoće.

Instrukcija MOV služi za pomicanje grafičkog kursora. Zapravo, ova instrukcija i na potpuno isti način kao i instrukcija PLOT, jedino što ne crta točku. Prema tome, u prethodnom primjeru mogli smo umjesto instrukcije PLOT staviti instrukciju MOV.

Ponovno naredbe POKE 522, mi možemo utjecati na način crtanja. (U naredbi POKE bit će više govora u narednom poglavlju.) Ako mi ima vrijednost 0, crtat će se bijelom bojom. To znači da ćemo na ekranu vidjeti ono što se crta ako je podloga crna. Ako mi ima vrijednost 1, crtat će se crnom bojom. To nam može poslužiti za crtanje po bijeloj podlozi ili za brisanje linija nacrtanih na crnoj podlozi. Ako mi ima vrijednost 255, crtat će se inverzno. To znači da će preko točke koja je nacrtana bijelom bojom biti stavljena crna točka, odnosno preko točke nacrtane crnom bojom biti stavljena bijela točka. Kad resetiramo računalo, vrijednost na adresi 522 je 0.

Kombinirajući ove načine crtanja, uz pravilnu upotrebu instrukcija za crtanje, možemo je na ekranu dobiti zaista impresivne grafike. Grafička rezolucija od 256 x x 256 točaka zadovoljava u većini slučajeva potrebe za prikazivanjem. Naredni program crta grafički funkcije sinus i kosinus u rasponu od 0 do 4°R. Upotrijebim instrukciju PLOT.

```
10 PRINT DIR$(12)
20 MOV 0,127
30 DRAW 255,127
```

```

40 FOR I=0 TO 255
50 A=127+126*SIN(6.28/128*A)
60 PLOT I,A
70 A=127+126*COS(6.28/128*A)
80 PLOT I,A
90 NEXT I

```

Vidimo da su grafikon prikazan pomoću niza točaka. Dolje rješenje dobiveno ako upotrijebimo instrukciju DRAW međutim tada moramo crtati jedan po jedan grafikon zbog toga što upotrebljavamo položaj grafičkog kursora. Zamijenite u prethodnom primjeru instrukcije PLOT instrukcijama DRAW - vidjet ćete da nije moguće crtati dva grafikona odjednom. Zbog toga u narednom primjeru svaki grafikon ima svoju petlju.

```

10 PRINT CLK*(12)
20 MOV 255,127
30 DRAW 0,127
40 FOR I=0 TO 255
50 A=127+126*SIN(6.28/128*A)
60 DRAW I,A
70 NEXT I
80 MOV 0,253
90 FOR I=0 TO 255

```

```

100 A=127+126*COS(6.28/128*A)
110 DRAW I,A
120 NEXT I

```

Vidimo da su tako nacrtane krivulje znatno homogenije jer ih instrukcija DRAW pravilno povezuje. Pogledajte npr. 80 u kojoj smo zbog crtanja drugog grafikona prvo poslali grafički kursor na početak grafikona kosinus-funkcije. Dodajte prethodnom primjeru sljedeću dvije linije i vidjet ćete kako možemo instrukcije DRAW i PLOT iskoristiti za brisanje po ekranu.

```

130 IF PEEK(522)=1 THEN POKE 52
2,0 : GOTO 20
140 POKE 522,1 : GOTO 10

```

MEMORIJA

U nekoliko navrata rekao sam da je memorija podijeljena na bajtove. Svak od ovih bajtova ima svoju adresu, odnosno jedan broj koji mu je pridješen i pomoću kojeg možemo pročitati ili upisati vrijednost u taj bajt. Procesor 6502 koji je ugrađen u mikračunalo «crab» adrese bajta čuva u dva bajta. Prema tome adresa pojedinog bajta jest binarni broj od šesnaest znamenki. Iz formule dva na šesnaest, možemo izračunati da mikroprocesor 6502 može adresirati 65536 bajtova odnosno $64 \cdot 1024$ bajta ili 64K. Neki od ovih bajtova nazivaju se u području memorije čiju vrijednost ne možemo mijenjati (ROM), a ostali su ili nepotrebni ili pokrivaju područjem memorije koje možemo mijenjati. Ne postoji nikakva opasnost da oštetimo računalo ni onda kada pokušamo promijeniti nešto u ROM-u ili na adresi na kojoj nema memorije.

Promjenu određene memorijske lokacije postizemo upotrebom naredbe PQKE. Iza ove naredbe slijede dva broja. Prvi označava adresu bajta u koji smjestimo vrijednost, a drugi vrijednost koja se smješta u taj bajt. Prvi broj mora biti u rasponu od 0 do 65535, a drugi od 0 do 255. Ako je bilo koji od ovih brojeva izvan predviđenog opsega računalo će prijaviti grešku.

Određene memorijske lokacije sadržavaju vrijednosti koje su od vitalnog značenja za uspješan rad BASICA. To su lokacije na adresama od 0 do 511 i njihov sadržaj nije preporučljivo mijenjati, osim ako točno znamo što

radimo. Na adresama iznad 1024 počinje BASIC program i u tom području mijenjanje vrijednosti pojedinih brojeva može izazvati promjenu u BASIC-u, što će kasnije dovesti do neispravnosti u radu programa. Ako želimo eksperimentirati s mijenjanjem sadržaja pojedinih memorijskih adresa, najbolje je da radimo prvenstveno memorije određene za BASIC-program pod pretpostavkom da nam BASIC ne dolazi do te lokacije.

Priključkom uključivanja računala «crab» nas pita

MEMORIJA ?

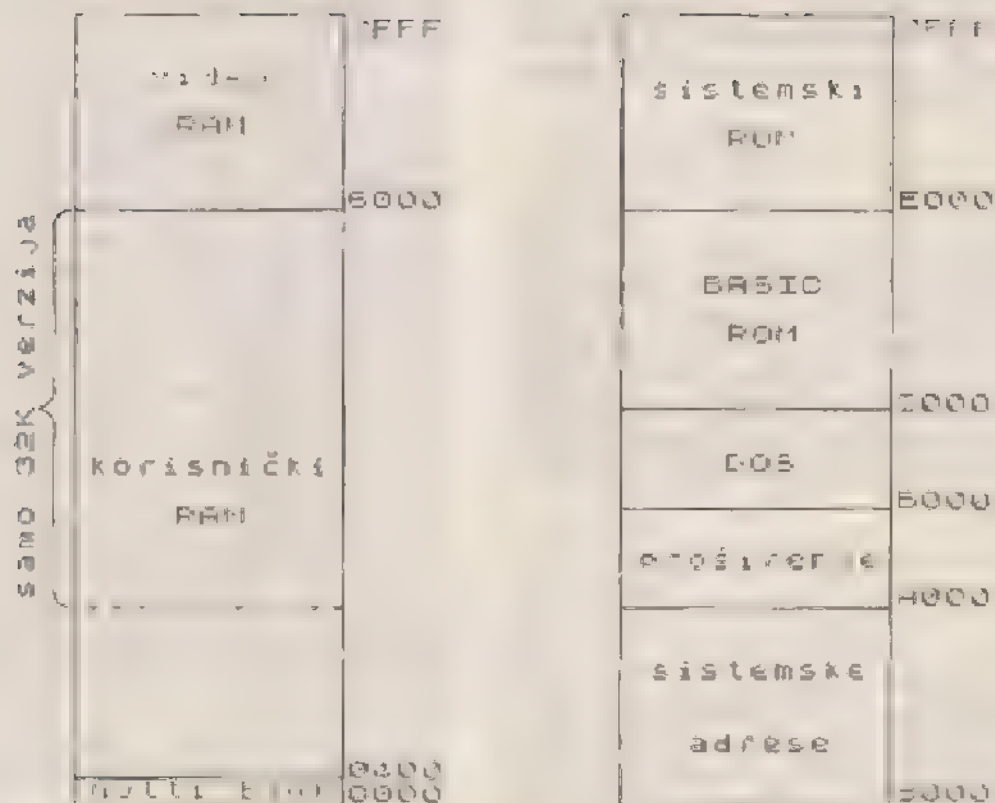
I tada možemo odgovoriti u koliko memorijskog prostora želimo rezervirati izvan područja dohvatljivog BASICA. Tako možemo rezervirati prostor na kojem će biti naša podaca i masinske rutine sigurno od opasnosti da ih prebriše BASIC.

Sadržaj pojedine memorijske lokacije možemo pročitati funkcijom PEEK. Za ovu funkciju vrijedi sve ono što je rečeno za numeričke funkcije. Argument ove funkcije koj se navodi u zagradu iza riječi PEEK označava adresu bajta koji čitamo i mora biti u rasponu od 0 do 65535. Naredba

PRINT PEEK (1000)

ispisat će sadržaj bajta koji ima adresu 1000. Za eks-

Slika 5



permaniranje s naredbama PEEK i POKE upotrijebit ćemo video-RAM. To je područje u memoriji u kojem «orao» čuva sliku. Ako izmijenimo bilo koju vrijednost u ovom području, to će odmah rezultirati promjenom na ekranu. Na primjer,

POKE 24007,255

orta crticu u gornjem desnom kutu ekrana. Svaki lock ekrana odgovara jedan bit u određenom bajtu. Osim točkica ekrana smješteno je u jedan bajt. Kada je bit koj pripada određenoj točkici postavljen na vrijednost 0, točka je crne boje. Ako je vrijednost bila postavljena na 1, točkica je bijele boje. Kada u jedan bajt memorije ekrana smjestimo vrijednost 255 koja binarno izgleda ovako:

11111111

upale se sve točkice koje odgovaraju tom bajtu. Zbog toga je u prethodnom primjeru nastala crtica.

Video memorija u mikroročunali «orao» nalazi se na adresama od 24576 do 32767, zauzima 8K. Ako pomnožimo 256 sa 256 (koliko točkica ima na ekranu), to podijelimo sa 8 (jer osam točaka zauzima jedan bajt) dobit ćemo točno 8K. Unesite ova primjer

10 FOR A=24576 TO 32768
20 POKE A,255
30 NEXT A

Vidimo da se slika popunjava odzgo prema dolje. U videomemoriji prvi bajt označava prvih osam točaka gornjeg kuta ekrana. Zatim slijed narednjih osam točaka i tako dalje sve dok se ne završi prvi red točkica na ekranu. Svakom redu točkica odgovara 32 bajta (32*8 = 256). Nakon toga slijede 32 bajta za drugi red točkica i tako dalje sve do završetka ekrana.

Instrukciju POKE možemo iskoristiti da definiramo vlastite znakove. Za potrebu definicije znakova «orao» posjeduje određene bajtove u kojima čuva vrijednost nula. Ako promijenimo vrijednost u tom bajtu, «orao» će to shvatiti kao da želimo predefinirati slova ili znakove. S obzirom na to da jedan bajt nije dovoljan da se čuva adresa nekog bajta u memoriji «orao» će pretpostaviti da je vrijednost koju mu damo prvi bajt adrese, a da je drugi bajt nula. Time smo ograničeni da možemo definiciju naših znakova staviti na adresu koja u binarnom obliku ima posljednjih osam znamenki nule. Dio memorije koji počinje od adrese koja u binarnom obliku završava sa osam nula i dakle 256 bajtova naziva se stranica (page) memorije. Zbog toga kažemo da se definicija naših znakova mora nalaziti u jednoj stranici memorije.

Uzeto ćemo za primjer stranicu memorije koja počinje na adresi 24064. Prvi bajt adrese za tu stranicu je 94

(sve vrijednosti dane su decimalno). Sada ćemo na tu adresu smjestiti osam bajtova koji će označavati definiciju znaka. Svakoj točki našeg znaka odgovara jedan bit u bajtu. Ako želimo da ta točka svijetli kada se znak iscrta, postaviti ćemo u bit vrijednost jedan. Predefinirali možemo samo 32 znaka odjednom. Zbog toga ćemo prvo u njima 10 do 30 postaviti u sve bajtove u vrijednost 255. Time će svi znakovi biti predefinirani u bijeli kvadratić. Sada možemo pristupiti definiciji našeg znaka. Definirali ćemo crni kvadratić s bijelim rubom. Naš znak možemo zamisliti kao matricu od 8 x 8 točaka. Gornji red točaka je prvi bajt u definiciji, drugi red točaka je drugi red i tako dalje. U prvom bajtu želimo da nam svijetle sve točke pa ćemo u njega smjestiti vrijednost 255 (11111111 binarno). U drugom redu želimo da nam svijetle prva i posljednja točkica pa ćemo staviti vrijednost 129 (10000001 binarno). Treći, četvrti, peti, šest i sedmi red isti su kao i drugi pa u njih idu vrijednost 129. U posljednjem redu ponovno želimo da svijetle sve točke pa zato stavljamo vrijednost 255.

Sada smo smjestili definiciju znaka u memoriju i još samo treba da kažemo «oru» koj dio znakova želimo predefinirati. Svakom dijelu znakova odgovara jedan bajt. U sljedećoj tablici vidite kojem dijeu slova odgovara koji bajt.

- 512 – korisnički definirani znakovi
- 513 – znakovi i brojevi
- 514 – velika slova
- 515 – mala slova

Predefinirali ćemo mala slova. Definiciju znakova smjestili smo na stranicu broj 94 i zato ćemo u bajt 515 koji odgovara malim slovima smjestiti vrijednosti 94. Ovo programa koji sve to obavlja.

```

10 FOR I=24064 TO 24319
20 POKE I,255
30 NEXT I
40 FOR I=24064 TO 24071
50 READ A
60 POKE I,A
70 NEXT I
80 POKE 515,94
90 DATA 255,129,129,129,129,129,129,129,255

```

Prvi znak u setu malih slova je strelica gore (posljednja tipka u drugom redu tastature). Pritiskom na tu tipku dobili ćemo znak koji smo definirali. Ako uključimo mala slova (tipka [PF1]) vidimo da bez obzira na to koje slovo pritisnemo dobivamo bijeli kvadratić.

Mala slova možemo «vratiti» ako u adresu 515 vratimo vrijednost nula (POKE 515,0) ili ako resetiramo računalo.

Vidjeli smo u prethodnoj tablici da adresa 512 odgovara korisnički definiranim znacima. To su znaci koji odgovaraju kodovima od 128 do 159. Ako nam unutar

nekog programa treba u konšističnu značku želimo zadržati sva slova i znakove, možemo definirati to područje. Jedni problemi je što ovakvi kreiranje znakove ne možemo unositi direktno s tastature, već ih moramo spisivati pomoću funkcije CHRS.

Ako definiramo mala slova, velika slova i znakove, možemo ih spisivati inverzno pritiskom na tipku (PF3).

Možemo definirati sve znakove, na primjer «orao» može raditi u editoru. Jedini je problem što gubimo mogućnost kretanja u eksternom editoru. Kad pritisnemo tipku (PF4) «orao» će prepoznati znak preko kojeg prazni kursor jedini, ako taj znak točno odgovara slovima koje on standardno posjeduje.

POZIVI STROJNIH POTPROGRAMA

Mikroprogram «orao» sadržava u sebi jedan velik strojni program koji omogućava sve ono što nam smo

prepoznali. BASIC-ovisti koji nemoguće da «orao» izvrši ono što smo od njega zahtijevali. Pojedini dijelovi tog programa mogu nam biti korisni, zbog toga je BASICU dodana još jedna funkcija. Ona je zanimljiva po tome što ne daje nikakav rezultat i nije važna koju vrijednost ima navedenu kao argument. Ona nam služi zato da pozovemo pojedine strojne potprograme. Bilo da pozivamo potprogram iz ROM-a ili rutinu koju smo sami napisali, postupat ćemo isto.

Prije moramo znati na kojoj se adresi nalazi naš potprogram. Adresu treba preračunati u dva bajta koje ćemo zatim smjestiti na adrese 11 i 12 decimalno. U bajt 11 ide niži bajt naše adrese ko, izračunamo po obrascu:

$$B=A-256*INT(A/256)$$

pri čemu je A adresa rutine koju želimo pozvati. U bajt 12 smjestit ćemo viši bajt adrese koju dobijemo ovako:

$$B=INT(A/256)$$

gdje je A adresa rutine koju pozivamo. Nakon toga pozvat ćemo potprogram tako da napišemo

$$U=USR(U)$$

Ova imamo našu funkciju za određivanje vrijednosti koja bit će već ovdje postavljamo isključivo radi pozivanja složenog potprograma. Pogledajmo to na primjer. U ROM-u se nalazi rutina koja omogućuje crtanje kružnice. Ona se nalazi na adresi 65286. U bajt 11 smjestimo čemo vrijednost B

B=65286-256*INT(65286/256)

a ona je šest. Dakle napisat ćemo

POKE 11,6

U bajt 12 smjestit ćemo vrijednost

B=INT(65286/256)

dakle vrijednost 255. To ćemo učiniti naredbom

POKE 12,255

Sada će funkcija USR pozivati rutinu z crtanje kružnice. No prije nego je pozovemo, moramo dati određene parametre. Parametre smjestimo na ove adrese:

226 – x koordinata središta
227 – y koordinata središta
248 – polumjer kružnice

Koordinate x i y jednake su kao i za naredbu PLOT. B i polumjer kružnice. Neka bit i raspon od 0-127. Pa ipak treba paziti da kružnica ne izlazi iz ekrana. Možemo slobodno iskusiti ono što ćemo u knjizi kružnica izlazi iz ekrana jer se tada jednostavno zatvaraju krivulje.

Sarita je zaista sve spremno za poziv potprograma za crtanje kružnice i možemo ga pozvati naredbom U=USR(U). Naredni program poziva ovu rutinu i crta nekoliko kružnica.

```
10 PRINT CHR$(12)
20 POKE 11,6
30 POKE 12,255
40 FOR I=10 TO 127 STEP 5
50 POKE 226,I
60 POKE 227,I
70 POKE 248,I-1
80 U=USR(U)
90 NEXT I
```

Potrebno je još napomenuti da rutine za crtanje kružnice primaju grafički kursor isto kao i ostale naredbe za crtanje. Nakon poziva ove rutine, grafički kursor ulazi sa na koordinatama koje smo naveli kao središte kružnice. Umjesto da smještamo koordinate središta s naredbama POKE, možemo upotrijebiti naredbu MOV. Nakon ove naredbe za središte kružnice uzet će se položaj grafičkog kursora. Preostaje nam samo da naredbom POKE upišemo poluprijer kružnice.

Upotrebom funkcijeUSR možemo zvesti programski prelazak iz BASICA u monitor bez potrebe da pritisćemo reset tipku. Dovoljno je da unesemo ovu sekvencu

```
10 POKE 11,167
20 POKE 12,255
30 U=USR(U)
```

Isto tako je moguće ponovno startati BASIC ovom sekvencom

```
10 POKE 11,17
20 POKE 12,221
30 U=USR(U)
```

MONITOR

Mikroračunalo "orao" opremljeno je završnim dodatkom koj nam omogućuje rad u strojnom jeziku. Čim uključimo računalo nalazimo se u monitoru. PROMPT monitora je zvjezdica i kad nju ugledamo, možemo se odlučiti za ulazak u BASIC prema već opisanom postupku. Isto tako možemo pozvati i BASIC bez umješavanja BASIC-programa, odnosno bez brisanja memorije. Ulazak u BASIC s brisanjem memorije naziva se hladni start BASICA, a ulazak bez brisanja memorije vrući start. Vrući start izvodimo naredbom 'BW.

Da bismo mogli vruće startati BASIC, potrebno je da je on već bio prethodno hladno startan, te da nisu oštećene vrijednosti na nultoj stranici.

Tipka reset vraća nas u monitor bez obzira na to kakav posao radi "orao". Zbog toga možemo ovom tipkom prekinuti čak beskonačne petlje u strojnom jeziku.

Namena je ove knjige da posluži kao priručnik za učenje BASICA. Zbog toga ovdje ne može biti mnogo govora o strojnom programiranju. Tekst koji bi poslužio kao minimalan uvod u strojno programiranje bio bi vjerojatno po obujmu znatno veći od ukupne dužine teksta u ovoj knjizi. Oni koji žele naučiti strojno programiranje moraju se poslužiti nekom drugom literaturom. Kako je «prao» građen oko mikrop procesora 6502 koji je vrlo raširen među mikračunalima, napisano je mnogo naslova koji se bave problematikom programiranja mikrop procesora 6502. Zbog toga pretpostavljam da svi zainteresirani neće imati problema pri nabavljanju literature. Oni koji su se sada prvi put sreli s računalom, za sada neka rad u strojnom programiranju ostave za bolja vremena. Oni slobodno mogu čitanje nastaviti od početka sljedećeg poglavlja.

Za rad u strojnom jeziku stoji nam na raspolaganju sedam monitorskih komandi: minassembler. Sve adrese koje se unose u monitor kao i sve vrijednosti koje unosimo moraju biti u heksadecimalnoj notaciji. Pogledajmo sada redom komande monitora.

Kao prvo, to je disassembler koji pozivamo komandom

***Xnnnn**

gdje nnnn označava heksadecimalnu adresu. Disas-

sembler je program koji pretvara kod strojnog programa u minemonike. Komandom X dobit ćemo 28 redova disassembliranog teksta. Ako želimo slijedeću stranicu dovoljno je da unesemo samo

***X**

bez adrese i disassembliranje će se nastaviti tamo gdje je prekinuto.

Za brzu pretragu memorije možemo uzeti naredbu

***Ennnn mmm**

gdje su nnnn početna adresa i mmmm završna adresa. Komanda E ispisival će nam heksadecimalne sadržaje batova počevši od adrese nnnn, završno s adresom mmmm. Ispisival će se osam vrijednosti u jednom redu. Ako želimo zaustaviti (ne prekinuti) ispisivanje možemo pritisnuti bilo koju tipku. Ponovnim pritiskom ispisivanje će se nastaviti. U naredbi E navođenje adresa je obavezno.

Čitanje i mijenjanje memorijske lokacije obavljamo pomoću komande

***Mnnnn**

U nastavku su navedene adrese nekih potprograma koji nam mogu biti korisni u našim strojnim programima.

- E500** – Potprogram za unos znaka s tastature. Potprogram vraća kod sushute tipke u adresi FC heksadecimalno.
- E71C** – Potprogram za unos znaka s tastature sa ispisom na ekranu. Ovaj potprogram istovjestan je prethodnom, jedino što se unoseni znak ispisuje na ekranu.
- E762** – Ispis znaka na ekran. Ovaj potprogram ispisuje na ekranu alfa-numerički znak uprkod se nalazi u akumulatoru.
- FFA7** – Ulazna točka monitora. Skokom na ovu adresu izvesti ćemo potpovnu inicijalizaciju monitora.
- E63B** – Ispis stringa na ekran. Ova potprogram omogućuje nam da ispisemo string koji je smješten u memoriji na ekran. Prije poziva potprograma potrebno je u registar Y smjestiti niz bajt adrese početka teksta, a u registar A viši bajt adrese. Obavezno je da string završava bajtom koji sadržava vrijednost 04.
- DD11** – Hardni start BASICA. Poziv ove adrese ima isti efekt kao i komanda *BC u monitoru.
- C274** – Vrući start BASICA. Skok na ovu adresu ima isti učinak kao i komanda *BW u monitoru.

- FE69** – Crtanje točke. Poziv ovog potprograma iscrtaće točku na ekranu isto kao instrukcija PLOT u BASICU. Prije nego pozovemo ovu rutinu moramo u okaciju hE2 smjestiti koordinatu x a hE3 u koordinatu y.
- FE8B** – Povlačenje linije. Na ovoj adresi nalazi se potprogram koji povlači liniju na ekranu. Prije nego ga pozovemo moramo ispuniti ove bajtove:
- E2 – x koordinata početka
 - E3 – y koordinata početka
 - E4 – x koordinata završetka
 - E5 – y koordinata završetka
- FF06** – Crtanje kružnice. O ovom potprogramu već je bilo govora u poglavlju o grafici. Ponovno samo da je prije poziva potrebno postaviti:
- E2 – x koordinata središta
 - E3 – y koordinata središta
 - F8 – polumjer kružnice

Programi u strojnom jeziku možemo spremati na vrpcu i to iz BASICA koji ćemo pozivati komandom *BW. Da bismo to mogli potrebno je prije nego počnemo s unošenjem strojnog programa prvo narediti start BASICA. Kada uđemo u BASIC program ćemo spremiti naredbom

SAVE "ime"nnnn,mmm

Ovdje su nnnn početna adresa našeg strojnog programa a mmmm njegova završna adresa. Ime je relativno ime koje podjeljujemo programu radi lakšeg snalaženja na kaseti. Sav ostali postupak jednak je kao i za spremanja BASIC programa. Program ćemo s vrpce učitat naredbom

LOAD "ime"nnnn

gdje je nnnn adresa na koju želimo da se program učita. Ime možemo izostaviti pa će biti učitani prvi program s vrpce. Ako izostavimo adresu na koju se program učitava, program će biti tretiran kao BASIC učitat će se na adresu 0400 heksadecimalno.

Sve adrese koje navodimo u naredbama SAVE i LOAD moraju biti HEKSADECIMALNE.

ŠTO DALJE?

Osnovna namjera pri radu na ovoj knjizi bila mi je da stvorim priručnik koji će pomoći početniku u prvom susretu s mikroracunalom. Zbog toga veći dio knjige govori o osnovnim BASIC-naredbama. činjenica je da neke BASIC-naredbe možda zaslužuju da se o njima više govori. međutim, ponavljam još jednom da se programiranje ne može naučiti ni iz stotinu knjiga. jedini pravi put da se svlada tehnika programiranja jest da se radi sa računalom. Želio bih da ova knjiga po tome bude vodič i pomoćnik.

Na kraju je potrebno istaknuti da je programski jezik BASIC namijenjen u prvom redu stjecanju prvih znanja o računalima, da bi svatko lako želio postati računarski osmišljena osoba morao svladati ovaj a potom prema svojim željama potrebama nek drug programski jezik. ima mnogo različitih teorija o tome koji bi programski jezik trebalo da bude "standard". Mislim da bi programski jezik PASCAL mogao zadovoljiti većinu zainteresiranih da svladaju još nek. vis. programski jezik. Za ozbiljniji rad na programima nije dovoljno samo poznavanje programskog jezika, potrebno je znati i mnogo detalja o samom stroju na kojem se radi. Oni koji žele izvući maksimum iz jednog mikroracunara, morat će svakako savladati strojni jezik.

Nadam se da će vam znanje koje ste stekli iz knjige biti dovoljan poticaj da se samostalno uputite u istraživanje svijeta mikroracunala.

DODACI

U prilogu knjige naći ćete četiri tablice. Prva je popis svih naredbi koje prepoznaje mikrolov BASIC. U toj tablici već je bilo govora o tekstu.

U tablici broj 2 nalazi se set znakova koji postoje u bilo kojim kodovima. Kodovi znakova pridruženi su prema standardu ASCII, s tim što su u njemu izmijenjeni zbog potrebe dodavanja naših slova.

Tablica broj 3 sadržava popis grešaka koje pojavljuje «orao» i njihovo tumačenje.

Na kraju je dodana i tablica naredbi za mikroprocesor 6502. Učinka tih naredbi na status registar u toj tablici mogu su označiti ovim kraticama:

N – NEGATIVE

Z – ZERO

C – CARRY

I – IRQ DISSABLE

D – DECIMAL MOD

V – OVERFLOW

Za označavanje učinka pojedine instrukcije upotrijebljeni su ovi znakovi:

* – flag postavjen prema rezultatu instrukcije

- – flag ne izmijenjen nakon instrukcije

1 – flag postavljen nakon instrukcije

0 – flag očišćen nakon instrukcije

x – flag bit 2 i 3 zadržava sedmi bit testirane memorije

y – flag zadržava šest bit testirane memorije

Na kraju knjige dodano je i neko ko BASIC programira.

TABLICA 1

~	EXP	POKE
*	FOR...NEXT	PUB
*	FREE	PRINT
	GOSUB...RETURN	READ
/	GOTO	REM
<	IF...GOSUB	RESTORE
<=	IF...GOTO	RIGHT\$
<>	IF...THEN	RND
*	INPUT	RUN
>	INT	SAVE

~	LEFT*	SUN	55	#	59	.	83	S	107	K
ABE	LEN	SUN	56	-	60	-	84	T	108	L
AND	LIST	SUN	57	%	61	=	85	U	109	le
ARE	LOAD	SUN	58	-	62	.	86	V	110	-
ATN	LOG	STD	59	-	63	.	87	W	111	-
CHRS	MDR	STR	60	-	64	-	88	-	112	-
CLAMP	MOV	TAB	61	-	65	A	89	-	113	-
CONT	NEW	TAN	62	-	66	-	90	-	114	-
LOU	NOT	USR	63	-	67	C	91	C	115	-
DATA	UN...GOSR	VAL	64	-	68	-	92	-	116	-
DEF FN	ON...GOTO		65	-	69	-	93	-	117	-
DIN	OR		66	-	70	-	94	-	118	-
DRAW	PEEK		67	-	71	-	95	-	119	-
END	PLOT		68	-	72	-	96	-	120	-

TABLICA 2

50	-	51	-	8	80	P	124	-	H	53	-	5	77	-	M	101	-	e	125	-	d	
52	-	57	-	9	81	-	Q	105	-	1	54	-	6	78	-	N	102	-	f	126	-	s
53	-	58	-	0	82	-	R	106	-	J	55	-	7	79	-	O	103	-	g	127	-	z

TABLICA 3

NF	NEXT bez prethodne naredbe FOR
SN	sintaktička greška
RG	RETURN bez prethodne naredbe GOSUB
NP	nedovoljno podataka, više podataka READ nego DATA
FP	vrijednosti izvan opsega (pri pozivu funkcije)
RO	rezultat izvan opsega (u matematičkim operacijama)
MP	memorija je puna ili ima previše GOSUB-nivoa
NN	skok ili poziv nedefinirane naredbe
PP	pogrešan poziv, pozvan nedefinirani element polja
DD	dvostruko dimenzionirana matrica
/0	dijeljenje s nulom
PN	nedozvoljena naredba upotrijebljena u komandnom modu
PT	pogrešan tip podataka (pri pozivu funkcije)
DN	prevelika dužina stringa
IK	izraz previše složen
ND	CONT bez prethodnog STOP ili (CTL) C
DF	nije definirana funkcija

TABLICA 4

		M	I	F	I	D	V
APC	A←A+M	*	*	*	*	*	*
AND	A←A AND M	*	*				
ASL		*	*	*	*		
BCD	if C=0 then PC=PC+ss		
DGS	if L=1 then PC=PC+ss	
BDQ	if Z=1 then PC=PC+ss	
BIF	A AND M, N ← M, V ← b5M	*	*	.	.	.	Y
BMI	if N=1 then PC=PC+ss	
BNE	if I=0 then PC=PC+ss	
BEI	if N=0 then PC=PC+ss	
BRV	PC ← M	
BVC	if V=0 then PC=PC+ss	
BVS	if V=1 then PC=PC+ss	
CLC	C←0	.	.	*	*	*	*
CLD	D←0	*	*
CLI	I←0	*	
CLV	V←0	*	*
CMP	A←M	*	*	*	*	*	*

PROGRAMI

PRERAČUNAVANJE BROJEVA

```

5 REM PRERAČUNAVANJE BROJEVA
10 GOSUB 850
20 POKE 534,0:POKE 535,229
30 DIM E$(15)
40 FOR Q=0 TO 15
50 READ E$(Q)
60 NEXT Q
70 PRINT CHR$(4);CHR$(10);CHR$(
10);CHR$(10)
80 PRINT "  D - dec    H - hex
   B - bin"
90 PRINT CHR$(10);CHR$(10);CHR$(
10);CHR$(10)
100 POKE 252,0
110 A=PEEK (252)
120 IF A=0 GOTO 110
130 IF A=66 GOTO 580
140 IF A=72 GOTO 420
150 IF A<>68 GOTO 100
160 PRINT CHR$(10); "Unesi dec.
broj";CHR$(10)
170 GOSUB 730

```

```

180 IF LEN(A$)=0 OR LEN(A$).5 G
OTO 170
190 FOR Q=1 TO LEN(A$)
200 IF MID$(A$,Q,1)<"0" OR MID$
(A$,Q,1)>"9" GOTO 170
210 NEXT Q
220 D=VAL (A$)
230 IF D>65535 GOTO 170
240 H=D
250 H$=""
260 B$=""
270 FOR Q=3 TO 8 STEP -1
280 P=INT (H/16^Q)
290 H=H-16^Q*P
300 B$=B$+E$(P)
310 P=P+4B
320 IF P>57 THEN P=P+7
330 H$=H$+CHR$(P)
340 NEXT Q
350 REM ISPIS REZULTATA
360 PRINT CHR$(13);CHR$(10);CHR
$(10);CHR$(10);CHR$(10);"DEC -
   ;D
370 PRINT CHR$(10); "HEX - ";H$
380 PRINT CHR$(10); "BIN - ";LE
FT$(B$,8);" ";RIGHT$(B$,8)
390 PRINT CHR$(10)

```

```

400 GOTO 70
410 REM POTPROGRAM ZA HEX
420 PRINT CHR$(10); "Unesi hex.
broj"; CHR$(10)
430 GOSUB 730
440 IF LEN(A$)=0 OR LEN(A$)>4 G
OTO 430
450 FOR Q=1 TO LEN (A$)
460 IF MID$(A$,Q,1) >="0" AND MID
$(A$,Q,1) <="9" GOTO 480
470 IF MID$(A$,Q,1) < "A" OR MID$
(A$,Q,1) > "F" GOTO 430
480 NEXT Q
490 D=0: FOR Q=1 TO LEN(A$)
500 P=ASC(MID$(A$,Q,1))
510 P=P-48
520 IF P>9 THEN P=P-7
530 P1=LEN(A$)-Q
540 D=D+P*16^P1
550 NEXT Q
560 GOTO 240
570 REM POTPROGRAM ZA BIN
580 PRINT CHR$(10); "Unesi bin.
broj"; CHR$(10)
590 GOSUB 730

```

```

600 IF LEN(A$)=0 OR LEN(A$)>16
GOTO 590
610 FOR Q=1 TO LEN(A$)
620 IF MID$(A$,Q,1) > "1" OR MID
$(A$,Q,1) < "0" GOTO 590
630 NEXT Q
640 D=0
650 FOR Q=1 TO LEN(A$)
660 P=VAL(MID$(A$,Q,1))
670 P1=LEN(A$)-Q
680 D=D+P*2^P1
690 NEXT Q
700 D=INT(D)
710 GOTO 240
720 REM ISPITIVANJE TASTATURE
730 PRINT CHR$(6);
740 A$=""
750 PRINT CHR$(13); A$; CHR$(5);
760 POKE 252,0
770 A=PEEK(252)
780 IF A=0 GOTO 770
790 IF A=13 THEN RETURN
800 IF A=8 AND LEN(A$)=1 THEN A
$=""
810 IF A=8 AND LEN(A$)>1 THEN A

```



```

$= LEFT$(A$,LEN(A$)-1)
820 IF A<40 OR A>70 GOTO 750
830 A$=A$+CHR$(A)
840 GOTO 750
850 PRINT CHR$(12)
860 PRINT CHR$(10);CHR$(10)
870 PRINT "      PRERAČUNAVANJE
PROJEVA";CHR$(10);CHR$(10)
880 PRINT "      Prije unos
enja broja"
890 PRINT "pritisnite D za deci
malni, H za"
900 PRINT "heksadecimalni ili B
za binarni"
910 PRINT "broj. Unesite broj i
pritisnite"
920 PRINT "(CR). Za novi broj
izaberite"
930 PRINT "ponovo D, H ili B."
940 PRINT CHR$(10);CHR$(10);CHR
$(10)
950 PRINT "PRITISNITE (CR)"
960 GOSUB 730
970 PRINT CHR$(12)
980 RETURN

```

```

990 REM PODALI
1000 DATA 0000,0001,0010,0011,0
100,0101,0110,0111
1010 DATA 1000,1001,1010,1011,1
100,1101,1110,1111

```

PROGRAM ZA CRTANJE

```

10 REM PROGRAM ZA CRTANJE
20 POKE 534,68: POKE 535,3
30 POKE 11,0:POKE 12,3
40 GOSUB 600
50 POKE 228,127
60 POKE 229,127
70 X1=127
80 Y1=127
90 M=2
100 POKE 522,255
110 X=PEEK(228)
120 Y=PEEK(229)
130 PLOT X,Y
140 U=USR(U)
150 A=PEEK(252)
160 PLOT X,Y
170 IF M=0 AND A=0 GOTO 100
180 POKE 522,M
190 IF A=32 THEN X1=X :Y1=Y : G
OTO 100
200 IF (A=48 OR A=57) AND M=2 T
HEN PLOT X,Y:GOTO 100
210 IF (A=48 OR A=57) AND M=0 GO
TO 100
220 IF A=61 THEN M=255
230 IF A=48 THEN M=2
240 IF A=49 THEN M=0
250 IF A=50 THEN M=1
260 POKE 252,0
270 IF A<52 THEN PRINT CHR$(7);
: GOTO 100
280 IF A=52 THEN MOV X1,Y1:DRAW
X,Y : GOTO 100
290 IF A=53 GOTO 340
300 IF A=54 GOTO 390
310 IF A=56 THEN PRINT CHR$(12)
:GOTO 50
320 IF A=55 GOTO 520
330 POKE 535,231:POKE 534,28:EN
D
340 MOV X1,Y1 : DRAW X1,Y
350 DRAW X,Y
360 DRAW X,Y1
370 DRAW X1,Y1
380 GOTO 490
390 L=ABS(X1-X)
400 H=ABS(Y1-Y)
410 R=SQR(L*L+H*H)
420 MOV X1,Y1
430 POKE 248,R
440 POKE 11,6
450 POKE 12,255

```

```

460 U=USR(U)
470 POKE 11,0
480 POKE 12,3
490 POKE 228,X
500 POKE 229,Y
510 GOTO 100
520 POKE 233,INT(X/8) : POKE 23
2,31-INT(Y/8)
530 PRINTCHR$(11);CHR$(10);
540 POKE 12,231
550 POKE 11,28
560 U=USR(U)
570 A=PEEK(252)
580 IF A=13 THEN POKE 12,3:POKE
11,0:GOTO 100
590 GOTO 560
600 PRINT CHR$(12)
610 FOR Q=768 TO 841
620 READ A
630 POKE Q,A
640 NEXT Q
650 DATA 32,176,229,144,35,201,
97,208,3,198,228,96,201
660 DATA 100,200,3,230,228,96,2
01,119,208,2,230,229,201
670 DATA 120,208,3,198,229,96,2

```

```

01,113,208,5,198,228,230
680 DATA 229,96,201,101,208,5,2
30,228,230,229,96,201,121
690 DATA 208,5,198,228,198,229,
96,201,99,208,251,230,228
700 DATA 198,229,96,32,176,229,
133,252,96
710 PRINT "      U programu kor
istimo dva"
720 PRINT "kursora. Vidljivi po
krećemo po"
730 PRINT "ekranu slovima Q,W,E
,A,D,Y,X,C."
740 PRINT "Nevidljivi kursor pr
itiskom na"
750 PRINT "razmaknicu saještamo
na vidlji-"
760 PRINT "vi kursor. Linija,
pravokutnik"
770 PRINT "i krug koriste oba k
ursora.      Ostale komande sui"
780 PRINT : PRINT "      0 - KRETA
NJE"
790 PRINT : PRINT "      1 - CRTAN
JE"
800 PRINT : PRINT "      2 - BRISA

```

```

NJE"
810 PRINT : PRINT "      3 - INVER
11RANJE"
820 PRINT : PRINT "      4 - LINIJ
A"
830 PRINT : PRINT "      5 - PRAVO
KUTNIK"
840 PRINT : PRINT "      6 - KRUG"

850 PRINT : PRINT "      7  PISAN
JE (<CR> za izlaz)"
860 PRINT : PRINT "      8 - BRISA
NJE EKRANA"
870 PRINT : PRINT "      9 - IZLAZ
IZ PROGRAMA"
880 PRINT:PRINT:PRINT "pritisni
(CR)";
890 POKE 252,0
900 A=PEEK(252)
910 IF A=13 THEN 900
920 PRINT CHR$(12)
930 RETURN

```

POREZ (IGRA)

```

10 GOSUB 9000
20 PRINT "    ZDRAVO!    Ja uzima
   za porez"
25 PRINT "brojeve s kojima je d
   jeljiv broj koji ti uzmeš."
30 CLEAR : GOSUB 810
40 GOSUB 120
50 GOSUB 330
60 GOSUB 420
70 IF N1=0 GOTO 100
80 GOSUB 530
90 IF N1>1 GOTO 40
100 GOSUB 690
110 GOTO 750
120 PRINT : PRINT
130 PRINT "Ti uzimaš? ";
140 INPUT K
150 K=INT(K)
160 IF K>0 AND K<=N GOTO 190
170 PRINT : PRINT "Taj broj nij
   e na listi!"
180 GOTO 130
190 IF L(K)=0 GOTO 170
200 IF K>1 GOTO 230
210 PRINT : PRINT "broj";k;"m
   ne daje za porez!"
220 GOTO 130
230 M=0
240 FOR I=1 TO K/2
250 IF K<>I*INT(K/I) OR L(I)=0
   GOTO 270
260 M=M+1
270 I(I)=1
280 I(I)=0
290 NEXT I
300 IF M=0 GOTO 210
310 I(I)=0
320 RETURN
330 Y=Y+K
340 PRINT : PRINT "za porez uz
   i mam";
350 FOR I=1 TO M
360 PRINT T(I);
370 Z=Z+T(I)
380 NEXT I
390 N1=N1+M+1
400 PRINT : PRINT "Moj iznos: ";
   Z;"Tvoj iznos: ";Y
410 RETURN
420 REM ISPIS LISTE
430 PRINT : PRINT "NOVA LISTA";

```

```

460 IF N1=0 GOTO 510
470 FOR I=1 TO N
480 IF L(I)=0 GOTO 500
490 PRINT I;
500 NEXT I
510 PRINT
520 RETURN
530 FOR I=N2 TO 1 STEP -1
540 IF L(I)=0 GOTO 620
550 FOR J=2 TO 1/2
560 IF L(J)=0 OR I<>J*INT(1/J)
GOTO 600
570 N2=I
580 M=1
590 RETURN
600 NEXT J
610 M1=M+1
620 NEXT J
630 PRINT : PRINT : PRINT "
Za poraz uzimam preostale,"
640 PRINT "zato što nema više b
rojeva koji"
645 PRINT "su djeljivi nekim od
preostalih brojeva."
650 I=I+M1
660 PRINT : PRINT "Moj iznos je

```

```

:";7
670 N1=0
680 RETURN
690 PRINT : PRINT "Ti imaš :";Y
700 IF Z>Y GOTO 730
710 PRINT : PRINT "Ti si POBJEO
JO!"
720 RETURN
730 PRINT : PRINT "JA SAM POBJE
DIO!"
740 RETURN
750 PRINT : PRINT "Želiš li nov
u igru?"
760 INPUT A$
780 IF LEFT$(A$,1)="-N" THEN END

790 PRINT CHR$(12);
800 GOTO 30
810 PRINT : PRINT "Koliko broje
va želiš na listi?"
820 INPUT N
830 N=INT(N)
840 N2=N
850 N1=N
860 IF N=0 GOTO 820
870 IF N<=50 GOTO 900

```



```

880 PRINT "Najviše 50 brojeva."
890 GOTO 810
900 DIM L(50)
910 DIM T(10)
920 Y=0
930 Z=0
940 M1=0
950 PRINT : PRINT "LISTA JE";
960 FOR I=1 TO N
970 PRINT I;
980 L(I)=1
990 NEXT I
1000 IF N>1 GOTO 1050
1010 PRINT : PRINT "Velikodušno
od tebe,"
1020 PRINT "sve za porez!"
1030 Z=1:Y=0
1040 GOSUB 690 : GOTO 750
1050 RETURN
1060 END
9000 PRINT CHR$(12);
9010 PRINT "                POREZ"
: PRINT
9020 PRINT "    Nakon što izaber
emo veličinu"
9030 PRINT "liste (broj brojeva
u igri od 1"
9040 PRINT "do 50), potrebno je
s te liste"
9050 PRINT "uzimati brojeve."
9060 PRINT "Brojeve uzimamo je
dan po jedan"
9070 PRINT "a svaki put kada u
zmemo jedan"
9080 PRINT "broj, 'orao' uzima
sve brojeve"
9090 PRINT "s kojima je taj br
oj djeljiv."
9100 PRINT "Nije dozvoljeno uze
ti broj koji"
9110 PRINT "nije djeljiv ni s j
ednim brojem"
9120 PRINT "od brojeva preostal
ih na listi."
9130 PRINT "Ako ostanu samo tak
vi brojevi,"
9140 PRINT "'orao' uzima sve
preostale."
9150 PRINT "Svi se brojevi k
oje uzmemo"
9160 PRINT "zbrajaju. Potrebno

```

je na kraju"

9170 PRINT "imati više nego što

je 'orao'."

9180 PRINT "uzeo za porez."

9190 PRINT : PRINT

9200 RETURN

GENERIRANJE ZVUKA

10 REM GENERIRANJE ZVUKA

20 REM UPIS STROJNOG PROGRAMA

30 FOR Q=8448 TO 8474

40 READ A: POKE Q,A

50 NEXT Q

60 REM POZIV STROJNOG PROGRAMA

70 POKE 11,0

80 POKE 12,33

90 PRINT CHR\$(12)

100 PRINT "PRITISNI BILO KOJU T
IPKU ZA KRAJ"

110 U=USR(U)

120 REM PODACI

130 DATA 160,0,136,208,8,141,0,
136,76,0,33,162,255,202,208

140 DATA 242,141,0,136,32,176,2
29,144,243,24,96,234

**Popularna naučno-tehnička biblioteka
KNJIGA DVADESET TREĆA**

Damir Muraja, dipl. inž.
-ORAO - UVOD U RAD I PROGRAMIRANJE-

Glavni urednik:
DUBRAVKO MALVIĆ

Tehnički urednik:
VJEKOSLAV BOSNAR

Recenzenti:
BORKO BORANIĆ, prof.
BRANKO KEREČIN, dipl. inž.

Lektor:
ZORKA HORVATIC

Priprema za tisak:
IZDAVAČKI ODJEL NARODNE TEHNIKE HRVATSKE
Zagreb, Dalmatinska 12

Tisak:
«Zadružna štampa» - OOUR Knjigotisak, Zagreb, Dalmatinska 5 i 12

